Parallel Algorithms for VLSI Channel Routing

by

NIMMAGADDA CHITTI RAJU

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KANPU

April, 1997

Parallel Algorithms for VLSI Channel Routing

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

by

Nimmagadda Chitti Raju

to the

Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur

April 1997

Certificate

Certified that the work contained in the thesis entitled "Parallel Algorithms for VLSI Channel Routing ", by Mr. *Nimmagadda Chitti Raju*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Dr. Sanjeev Saxena,
Assosciate Professor,
Dept. of CSE,
IIT Kanpur.

April 1997

to
my parents

Acknowledgements

I would like to express my sincere gratitude to Dr. Sanjeev Saxena, for his invaluable guidance and constant encouragement throughout the work. It has been a great pleasure working under him, who from the beginning has devoted a great deal of time towards the completion of this thesis.

Special thanks to naidu, gvrk, kala, veluru, gsri, siva reddy, jagan and all my friends of MTech'95 batch, for their constant assistance and also for giving memorable company during my stay at IIT Kanpur.

I do acknowledge the constant support and encouragement from my parents and my sisters. It was their support, that gave me the spirit to carry on my study at IIT Kanpur.

Abstract

In this thesis parallel algorithms for channel routing in River routing model, Knock-Knee model, and Manhattan diagonal model, are proposed. In River routing model, an $O(\log n)$ time optimal parallel algorithm with $\frac{n+k}{\log n}$ processors is obtained on a CREW model. A constant time algorithm is also obtained, using $O(n^2)$ processors on a COMMON CRCW model. In Knock-Knee model, two optimal $O(\log l)$ time parallel algorithms for two-terminal nets are proposed; the algorithms uses $2d-1$ tracks to route the nets, and can be implemented on a CREW model. An $O(\log l)$ time parallel algorithm is obtained for multi-terminal nets; the algorithm in the worst case uses $4d-1$ tracks to route the nets. In Manhattan diagonal model an $O(\log l)$ time optimal parallel algorithm for two-terminal nets is obtained; the algorithm routes the nets in $d+1$ tracks, and it can be implemented on a CREW model.

Contents

List of Figures

Chapter 1

Introduction

Channel routing plays an important role in the development of automated layout systems for integrated circuits[8]. A *channel* is the area between two parallel lines, called *sides*. Along the lines are numbers called *terminals*, and terminals with the same number form a *net*. Channel routing is the problem of connecting all the terminals of each net, such that no two nets are electrically shorted. The goal is to minimize the distance between the parallel lines, called *width* of the channel. Fig 1.1 shows an example channel routing problem.

A net may contain any number of terminals. A net with q terminals is called *q-terminal* net. A net with one terminal is not interesting, because it does not require any routing. A net with two terminals is called *two-terminal* net, and a net with more than two terminals is called a *multi-terminal* net. Terminals on the upper side are called *top* terminals, and terminals on the lower side are called *bottom* terminals.

The channel consists of rectilinear grid of horizontal and vertical lines. The horizontal lines are called *tracks*, and the vertical lines are called *columns*. Both sides of the channel lie on tracks, and the terminals reside at grid points. The distance between the sides is called *width* of the channel. The length of the sides is called *length* of the channel.

There are two primary ways to represent the channel routing problem. The first, called *column list*, simply enumerates the numbers for each column in order from left to right, for each side of the channel; a 0 represents that the terminal is an empty terminal, and it does not belong to any net. The second, called *net list*, comprises

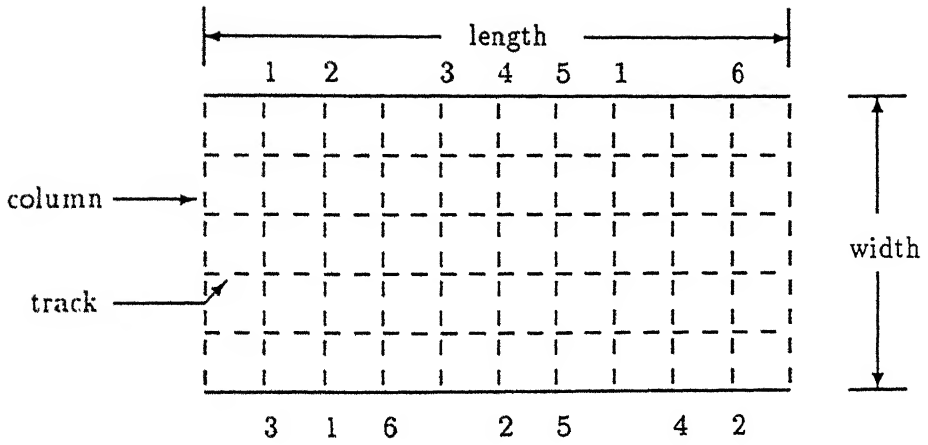


Figure 1.1: A channel routing problem

an enumeration of the nets, each represented as two sets of top terminals, and bottom terminals. The channel routing problem shown in Fig 1.1 is represented in *column list* as

UpperSide	0	1	2	0	3	4	5	1	0	6	0
LowerSide	0	3	1	6	0	2	5	0	4	2	0

The same channel routing problem is represented in *net list* as

Net	*TopTerminals*	*BottomTerminals*
1	2, 8	3
2	3	6, 10
3	5	2
4	6	9
5	7	7
6	10	4

Through out the thesis, we assume that the channel routing problem is represented as *net list*.

As we are at the limits of what can be achieved through sequential computation, parallel computation models are emerged to cope up with our needs. To make use of

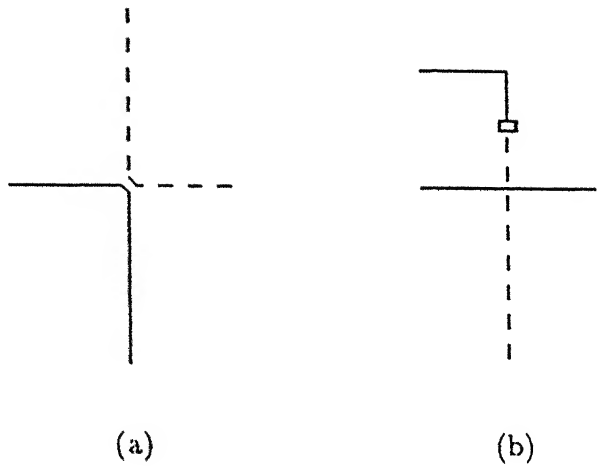


Figure 1.2: a) Knock-Knee: Sharing a corner. b) Contact cut

these parallel computation models, one has to develop new algorithms by exploiting the inherent parallelism in the problems. In this thesis, we have developed parallel algorithms for channel routing problem.

1.1 Routing Models

A variety of models have been proposed for channel routing[11], with differences depending on the number of layers available, and the ways in which wires are allowed to interact. Some of the commonly used models are River routing model[17], Knock-Knee model[14], and Manhattan diagonal model[5].

1. *River Routing Model:* This model uses only one layer to route the nets. Obviously, this model can be used only when the labels of the terminals are in the same order on both sides of the channel.

2. *Knock-Knee Model:* In this model wires are allowed to share corners, but are not allowed to overlap, as shown in Fig 1.2(a). The model uses two layers to route the nets. Whenever a net changes from one layer to another layer a *contact cut* will be used, as shown in Fig 1.2(b). The *density* of the channel in this model is defined as the maximum number of nets crossing any vertical cut of the channel.

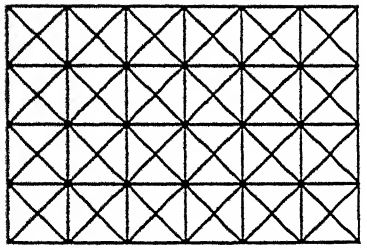


Figure 1.3: Channel in Manhattan Diagonal model

3. Manhattan Diagonal Model: In this model nets can be routed vertically, horizontally, and diagonally(see Fig 1.3). The model uses two layers to route the nets. Vertical wires are routed in one layer, horizontal and diagonal wires are routed in the other layer. The *density* of the channel in this model is defined as the maximum number of nets crossing or touching any vertical cut of the channel.

1.2 Overview of the Thesis

In Chapter 2 parallel computational models, and some standard parallel algorithms are presented. In Chapter 3, channel routing in River routing model is discussed. An optimal $O(\log n)$ time parallel algorithm, and a constant time parallel algorithm are proposed in this chapter. In Chapter 4, channel routing in Knock-Knee model is discussed. Two optimal $O(\log l)$ time parallel algorithms for routing of two-terminal nets are proposed in this chapter, where l is the length of the channel; the algorithms uses $2d-1$ tracks to route the nets, where d is the density of the channel. An $O(\log l)$ time parallel algorithm is also proposed for routing of multi-terminal nets; the algorithm in the worst case uses $4d-1$ tracks to route the nets. In Chapter 5, channel routing in Manhattan diagonal model is discussed. An $O(\log l)$ time optimal parallel algorithm for routing of two-terminal nets is proposed; the algorithm uses $d+1$ tracks to route the nets. Finally conclusions are in Chapter 6.

Chapter 2

Parallel Processing Preliminaries

In this thesis we will be using SIMD shared memory model, in which all processors execute the same program, and communicate by reading and writing in shared memory. In Exclusive Read Exclusive Write (EREW) model, no two processors are allowed to read from or write into the same location. In Concurrent Read Exclusive Write (CREW) model, more than one processor can read from the same location. In Concurrent Read Concurrent Write (CRCW) model, more than one processor can read from and write into the same location. In this thesis we will be using the COMMON CRCW model only, in which all processors writing to a given memory location must attempt to write the same value, which then gets stored in the memory location.

If the worst-case running time of the best known sequential algorithm for a problem is t, an *optimal* parallel algorithm for the same problem runs in $\frac{t}{p}$ time using p processors.

A model is said to be *self-simulating*, if an algorithm which takes $O(t)$ time with p processors, can also be implemented on that model in $O(rt)$ time with $\frac{p}{r}$ processors($r > 1$). EREW, CREW, COMMON CRCW models are self-simulating models.

We next discuss some of the standard parallel algorithms we are using.

- The prefix computation problem is to compute all initial products $a_0 * a_1 * \ldots * a_i$, $i = 1, 2, \ldots, n$, of a set of n elements, where $*$ is an associative operation. This problem can be solved in $O(\log n)$ time with $\frac{n}{\log n}$ processors on EREW model [10].

- Given an array $A[1..n]$, the *nearest larger* on the right for $A[i]$ is $A[j]$, if $A[i] < A[j]$ and $A[i] \geq A[k]$, for $i < k < j$. Given an array $A[1..n]$, the *nearest smaller*

on the right for $A[i]$ is $A[j]$, if $A[i] > A[j]$, and $A[i] \leq A[k]$, for $i < k < j$. Given an array of n items, we can find the *nearest larger* and *nearest smaller* on the right for each element in $O(\log n)$ time with $\frac{n}{\log n}$ processors on CREW model[4].

- Given two sorted arrays $A[0..n-1]$ and $B[0..n-1]$, the arrays can be merged in $O(\log \log n)$ time using $\frac{n}{\log \log n}$ processors on the CREW model[9].

- The linked list ranking problem is, given a linked list find the number of items in the linked list after it. Linked list ranking problem can be solved optimally in $O(\log n)$ time on EREW model[1].

We will be using prefix-sums routine to compact an array. Let us assume that an array of size n contains only m items ($m < n$), then prefix-sums routine can be used to compact the items; basically an auxiliary array is used, i^{th} entry is 0 if i^{th} location is empty and 1 if non-empty; prefix-sums on auxiliary array will give new position of the items in compact order.

Given a binary array $A[1..n]$, then index of the nearest 1 on the left and right can be found by using prefix-maxima routine. Basically an auxiliary array B is used. If $A[i] = 1$ then $B[i] = i$; else $B[i] = 0$. Compute prefix-maxima for B. The $(i-1)^{th}$ prefix-maxima will give the index of the nearest 1 on the left for $A[i]$. If $A[i] = 1$, then $C[n-i] = n-i$; else $C[n-i] = 0$. Compute prefix-maxima for C. The $(i-1)^{th}$ prefix-maxima will give the index of the nearest 1 on the right for $A[n-i]$. Prefix-maxima of n numbers in the range $1..n$ can be found in $O(1)$ time with $n \log^2 n$ processors, on COMMON CRCW model.

Given two sorted arrays $A[0..n-1]$ and $B[0..n-1]$, location of $A[i]$ in the array B can be found by merging the arrays A and B. Let k be the index of $A[i]$ in the merged array, then $k - i$ is the index of $A[i]$ in the array B. This problem can also be solved in $O(1)$ time using n^2 processors, on CREW model. Assign n processors to each item in A. Basically an auxiliary array $C[1..n][1..n]$ is used. Put 1 in $C[i][j]$ if $A[i] \geq B[j]$. Otherwise 0 is put. Since the array B is sorted, there is exactly one transition from 0 to 1, in each row of C. The index j, for which $C[i][j-1] = 0$ and $C[i][j] = 1$ is the index of $A[i]$ in B.

Let G be a graph with n nodes, containing a set of cycles(circular linked lists), and

each node is associated with a value. The node having maximum value in each cycle can be found in $O(\log n)$ time with $\frac{n}{\log n}$ processors on EREW model. Basically algorithm of Anderson and Miller[1] is used. At first each node will set, itself as the node having maximum value seen so far by it. Let p be the predecessor of node q, while q is being removed. If the value of q is larger than value of p, then q will set the value of p as value of q, and the node having maximum value seen so far by node p as the node seen by q. When ever an isolated node is removed from the cycle, and the node is the only node in the cycle, then the node having maximum value seen by it, is indeed the node having maximum value in the cycle. After $\log n$ iterations we will be left with at most $\frac{n}{\log n}$ nodes. The node having maximum value in each cycle can be found in $O(\log n)$ time with $\frac{n}{\log n}$ processors, using pointer jumping technique[9].

Chapter 3

River Routing Model

In this chapter we present parallel algorithms for channel routing in River routing model. The parallel algorithms are obtained by parallelizing the algorithm of Dolev et. al[6]. Even though Dolev et. al[6] claimed that their algorithm can be implemented in $O(n^2)$ time, it is easy to see that their algorithm in fact takes $O(n+k)$ time, where n is the number of nets and k is number of jogs in the layout(whenever a net moves from one track to another track, it will result in a jog). We assume that the nets are in the sorted order, based on the position of their terminals.

We obtain an $O(\log n)$ time optimal parallel algorithm for channel routing in River routing model, using $\frac{n+k}{\log n}$ processors, on the CREW model. We also obtain a constant time algorithm for the same problem using $O(n^2)$ processors, on the COMMON CRCW model.

This chapter is organized as follows. In Section 3.1, the serial algorithm of Dolev et. al[6] is presented. In Section 3.2 an optimal $O(\log n)$ time parallel algorithm is presented. In Section 3.3, an example for the optimal algorithm is given. Finally in Section 3.4, a constant time algorithm is presented.

3.1 Serial Algorithm

Let the given nets be $N_1 \ldots N_n$; and let Q_i be the top terminal and P_i be the bottom terminal of net N_i. The problem is partitioned into blocks. A *left block* is a maximal

sequence of nets $N_i, \ldots, N_j$ such that $i \leq k \leq j$

1. $P_k > Q_k$ and

2. $P_k \geq Q_{k+1}$ if $k < j$.

Similarly the right block is defined as follows: A *right block* is a maximal sequence of nets $N_i, \ldots, N_j$ such that $i \leq k \leq j$

1. $Q_k > P_k$ and

2. $Q_k \geq P_{k+1}$ if $k < j$.

A net with $P_i = Q_i$ is called trivial net, and is routed directly using a vertical wire connecting its terminals.

Since there is no interaction between any two blocks, once the problem is decomposed into blocks, each block can be routed independently. We now consider the routing of a left block.

Consider two nets i and j, such that $j > i$. If at any point $r \leq P_i + j - i$, the wire for net j comes less than $j - i + 1$ tracks from channel bottom, then it will not possible to route the nets i to $j - 1$ (maintaining unit separation). Thus about each terminal P_i on the channel bottom we draw a collection of forbidden regions to indicate the region from which net j must be excluded, for $j > i$. The Fig 3.1 shows the forbidden regions about a net i.

The forbidden region about net j, does not restrict net i, if $P_j + i - j < Q_i$, as the forbidden region is to the left of Q_i, and a net in a left block is always routed to the right of Q_i.

While routing a net i, its forbidden regions about net j; $j < i$, for which $P_j = P_{j-1}+1$ are not considered. This is due to the fact that the forbidden regions due to these nets are hidden in the forbidden region due to its previous net k having $P_k > P_{k-1} + 1$.

Each net i in the block can be routed as follows. First the net is routed vertically downwards from Q_i until it meets the forbidden region about net j. Then it is routed along the forbidden region about net j, until it meets the forbidden region of net k. This process is repeated until the terminal P_i is reached. The switching from one forbidden region to another results in a jog. The Fig 3.2 shows the routing of net 4. The dotted

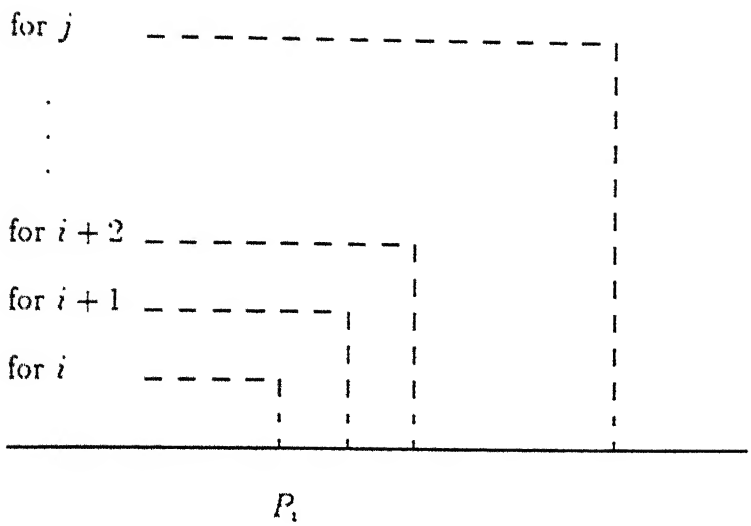


Figure 3.1: Forbidden Regions

lines represent the forbidden regions and the solid line with arrows represents the routing of the net.

Nets in the right block are routed as follows. We construct a new problem instance by interchanging the P_i and Q_i for each net in the right block. The solution of the original problem is the mirror image of the new problem.

Although Dolev et. al[6] claim that their sequential algorithm takes $O(n^2)$ time, it is easy to see that their algorithm in fact takes $O(n+k)$ time, where k is number of jogs in the layout. In the worst case there can be $O(n^2)$ jogs.

3.2 Optimal $O(\log n)$ Time Algorithm

In this section we describe an optimal $O(\log n)$ time parallel algorithm for River routing, on CREW model. There is no need to explicitly divide the problem into blocks. To understand the solution easily, the problem is divided into blocks, in the previous section. A net i is said to be a left net, if $P_i > Q_i$. A net is said to be a right net, if $Q_i > P_i$.

1. If the net is a right net, convert it into a left net.

   Remark: Interchange the terminals of a net i, if it is a right net. This can be done in $O(1)$ time using n processors or in $O(\log n)$ time with $\frac{n}{\log n}$ processors on

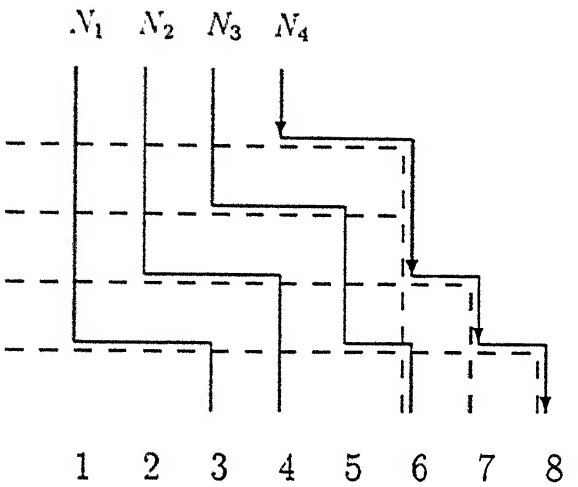


Figure 3.2: Routing of a net through forbidden regions

EREW model, hence also on CREW model.

2. Find all nets N_i, such that forbidden regions about N_i are effective.

   Remark: If $P_i > P_{i-1} + 1$, then forbidden regions about net i are effective; otherwise the forbidden regions about net i are hidden in forbidden regions about net $i-1$. First net is always assumed to be effective. An auxiliary array will be used. Put 1 in the ith location if net i is effective, otherwise 0. Compact this array using prefix-sums. Let the new array be P'.

3. For each net i find the first net j, such that $Q_i - i < P'_j - j$.

   Remark: Compute $Q_i - i$ and $P'_j - j$, then merge these arrays. The difference between the index of net i in the original array and the merged array, will give the index of net j. This step gives the first net j,which effects the routing of net i. This step can be done in $O(\log n)$ time using $\frac{n}{\log n}$ processors on CREW model.

4. For each net i find the last net j, in the set of nets obtained in the Step 2, such that $P_i \geq P'_j$.

   Remark: This step gives the last net j, which effects the routing of net i. The prefix-sum at i^{th} location in Step 2 gives the index of net j.

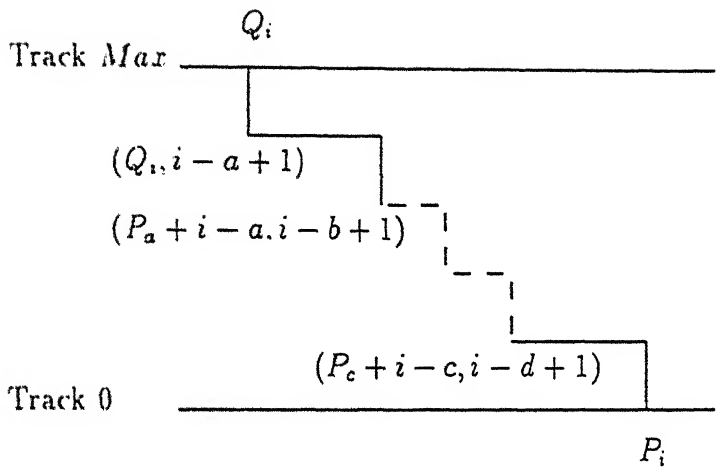


Figure 3.3: Routing of a left net

By the end of this step, for each net i, we know the forbidden regions about first net l and last net m, which effects the routing of net N_i. The nets between l and m in the array obtained in Step 2 gives the other nets which effects the routing of net i. The difference between the index of net N_l and the index of net N_m in the array obtained in Step 2, gives the number of jogs for net N_i.

5. Find the number of tracks required to realize the layout.

   Remark: For each net find the first track, in which the net should be routed. Let j be the first net, which effects the routing of net i, then $i-j+1$ is the required track for net i. Find the maximum of all tracks. Maximum of n numbers can be found in $O(\log n)$ time with $\frac{n}{\log n}$ processors on CREW model.

6. The realization of the layout of a net i is obtained as follows. Let us assume that forbidden regions about nets $a, b, \ldots, c, d$ effects the routing of net i. Fig 3.3 shows the routing, if the net is a left net. Fig 3.4 shows the routing, if the net is a right net.

   Remark: For realization each net will be given processors equal to the number of jogs(which is same as number of nets effecting it). Let k_i be the number of processors required by net i. Then $k = \sum_{i=1}^{n} k_i$ will give the total number of processors required. Sum of n numbers can be found in $\frac{n}{\log n}$ time on CREW

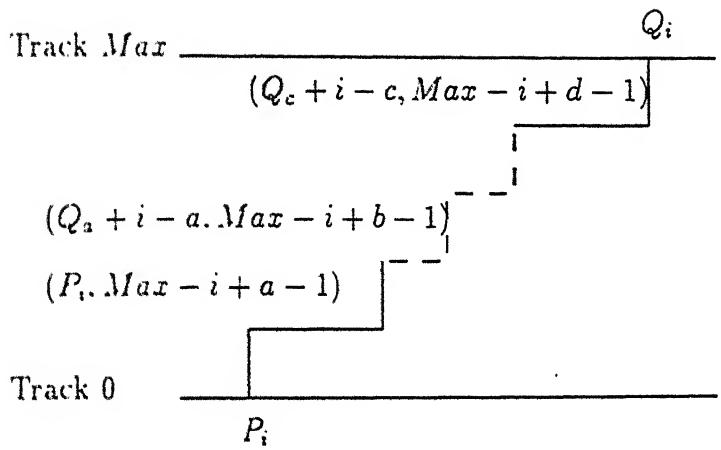


Figure 3.4: Routing of a right net

model. Net i will assign one processor to each net which effects it. Let us assume that net j is effective for net i. Let l be the net immediately to the left of net j, and m be the net immediately to the right of net j. Then the processor of net j, will realize the jog $(P_i + i - l, i - j + 1), (P_j + i - j, i - j + 1), (P_j + i - j, i - m + 1)$, if the net i is a left net, or jog $(Q_l + i - l, Max - i + j - 1), (Q_j + i - j, Max - i + j - 1), (Q_j + i - j, Max - i + m - 1)$, if the net is a right net. If the net l is not effective for net i, then the first co-ordinate in the jog will be $(Q_i, i - j + 1)$, if the net i is a left net; else the first co-ordinate will be $(P_i, Max - i + j - 1)$, if the net is a right net. Similarly if the net m is not effective for net i, then the last co-ordinate in the jog will be $(P_i, 0)$, if the net i is a left net; else the last co-ordinate will be (Q_i, Max), if the net is a right net. This can be done in $O(1)$ time with k processors, hence in $O(\log n)$ time with $\frac{k}{\log n}$ processors, where k is total number of jogs in the layout. If i is a trivial net, then realize the vertical connection $(P_i, 0), (Q_i, Max)$; else let j be the first net effecting the net i, then realize the vertical connection $(Q_i, Max), (Q_i, i - j + 1)$, if the net is a left net; else realize the vertical connection $(P_i, 0), (P_i, Max - i + j - 1)$, if the net is a right net. This can be done in $O(\log n)$ time with $\frac{n}{\log n}$ processors.

All the steps in the algorithm can be done in $O(\log n)$ time. The maximum number of processors used at any step is at most $\frac{n+k}{\log n}$. Hence the following theorem:

Theorem 3.1 *Channel routing problem in River routing model can be solved optimally in $O(\log n)$ time with $\frac{n+k}{\log n}$ processors on CREW model.* ∎

3.3 Example

In this section we will give an example for the algorithm discussed in the previous section. Let the given nets be

Net	1	2	3	4	5	6	7	8
Q_i	1	2	3	5	7	9	11	12
P_i	2	3	5	6	7	8	9	11

1. Convert right nets into left nets. Here 6, 7, and 8 are right nets. The new problem is

Net	1	2	3	4	5	6	7	8
Q_i	1	2	3	5	7	8	9	11
P_i	2	3	5	6	7	9	11	12

2. Find all nets i, such that forbidden regions about net i are effective.

1	0	1	0	0	1	1	0

The prefix-sums are

1	1	2	2	2	3	4	4

The compacted array of effective nets is

$P' =$

1	3	6	7

3. Compute $Q_i - i$, and $P'_i - i$.

$Q_i - i =$

0	0	0	1	2	2	2	3

$P'_j - j =$

1	2	3	4

For each net i find first net j such that $Q_i - i < P'_j - j$.

1	1	1	3	6	6	6	7

4. For each net i, find last net j such that $P_i \leq P'_j$.

1	1	3	3	3	6	7	7

5. Find the number of tracks required to realize the layout.

1	2	3	2	0	1	2	2

The maximum value is 3. Hence 3 tracks are required to realize the layout.

6. Realize the layout(See Fig 3.5).

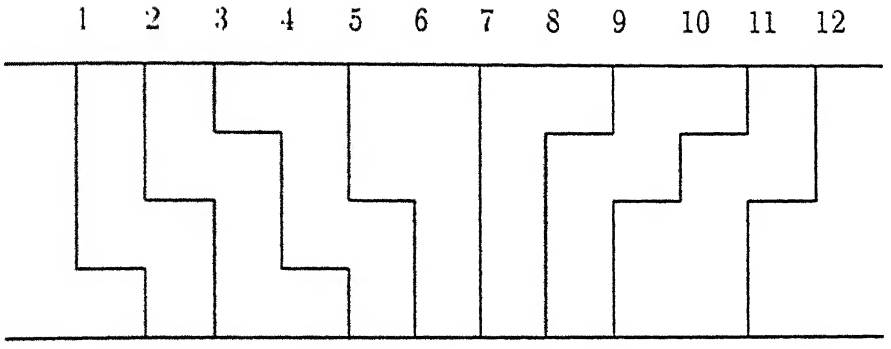


Figure 3.5: The layout for the example

3.4 Constant Time Algorithm

In this section we describe a constant time algorithm with n^2 processors, on COMMON CRCW model. This algorithm at the high level differs from the previous algorithm in Step 2. Instead of compacting, for each effective net, find the nearest effective net on the left and right.

1. If the net is a right net convert it into a left net.

2. Find all the nets such that forbidden regions about it are effective. For each effective net, find the nearest effective net on the left and right.

   Remark: Forbidden regions about a net i are effective if $P_i > P_{i-1} + 1$. First net is always assumed to be effective. An auxiliary array will be used. Put 1 in the ith location if net i is effective, otherwise 0. For each 1, find the location of the nearest 1 on the left and right. Use prefix-maxima routine, as explained in Chapter 2 to solve this. Prefix-maxima of n numbers in the range $1..n$ can be found in $O(1)$ time using $n \log^2 n$ processors, on COMMON CRCW model.

3. For each net i find the first net j, such that $Q_i - i < P_j - j$.

   Remark: Given two sorted arrays $A[0..n-1]$ and $B[0..n-1]$, location of $A[i]$ in the array B can be found in $O(1)$ time using n^2 processors, on CRCW model.

4. Find the number of tracks required to realize the layout.

Remark: For each net find the first track, in which the net should be routed. Let j be the first net, which effects the routing of net i, then $i-j+1$ is the required track. Find the maximum of all tracks. Maximum of n numbers can be found in $O(1)$ time with n^2 processors, on CRCW model.

5. Realize the layout.

   Remark: For realization each net i will be given n processors, (assign one processor to each net). Let a be the first net, which effects the routing of net i. If $j < a$ or $P_j > P_i$ or net j is not effective, then processor j sits idle. Otherwise processor j realizes the jog $(P_l+i-l, i-j+1), (P_j+i-j, i-j+1), (P_j+i-j, i-m+1)$, if the net i is a left net, else realizes the jog $(Q_l+i-l, Max-i+j-1), (Q_j+i-j, Max-i+j-1), (Q_j+i-j, Max-i+m-1)$, if the net is a right net; where l is the effective net immediately to the left of j, and m is the effective net immediately to the right of j. If the net l is not effective for net i, then the first co-ordinate in the jog will be $(Q_i, i-j+1)$, if the net i is a left net; else the first co-ordinate will be $(P_i, Max-i+j-1)$, if the net is a right net. Similarly if the net m is not effective for net i, then the last co-ordinate in the jog will be $(P_i, 0)$, if the net i is a left net; else the last co-ordinate will be (Q_i, Max), if the net is a right net. This can be done in $O(1)$ time with n^2 processors, on CRCW model. If i is a trivial net, then realize the vertical connection $(P_i, 0), (Q_i, Max)$; else let a be the first net effecting the net i, then realize the vertical connection $(Q_i, Max), (Q_i, i-a+1)$, if the net is a left net; else realize the vertical connection $(P_i, 0), (P_i, Max-i+a-1)$, if the net is a right net. This can be done in $O(1)$ time with n processors.

All the steps in the algorithm can be done in $O(1)$ time. The maximum number of processors used at any step is at most n^2. Hence the following theorem:

Theorem 3.2 *Channel routing problem in River routing model can be solved in constant time with n^2 processors on COMMON CRCW model.* ∎

Chapter 4

Knock-Knee Model

In this chapter we describe the channel routing in Knock-Knee model. This model is proposed by Rivest et. al[14]. The model uses two layers to route the nets. In this model wires are allowed to share corners, but are not allowed to overlap. Channel routing in Knock-Knee model using minimum number of tracks is NP-complete[15]. However there exist algorithms that can route any two-terminal channel routing problem in polynomial time using $2d-1$ tracks, where d is the density of the channel[3, 14]. The *density* of the channel routing problem is the maximum number of distinct nets crossing any vertical cut of the channel. There are nets for which $2d-1$ tracks are required; thus $2d-1$ is the worst case lower bound for two-terminal channel routing problem in Knock-Knee model[12]. For multi-terminal nets $3d+O(\sqrt{(d \log d)}$ is the best known upper bound[7].

In this chapter the algorithm of Berger et. al [3] is parallelized to give an optimal $O(\log l)$ time parallel algorithm, that can route any number of two-two-terminal nets in $2d-1$ tracks. Later the algorithm of Berger et. al[3] is modified to give another $O(\log l)$ time optimal parallel algorithm, that can route the two-terminal nets using $2d-1$ tracks. We also obtain an $O(\log l)$ time parallel algorithm for routing of multi-terminal nets. The algorithm in the worst case uses $4d-1$ tracks. All the algorithms presented in this chapter can be implemented on CREW model.

This chapter is organized as follows. In Section 4.1, routing of two-terminal nets is discussed. In Section 4.2, routing of multi-terminal nets is discussed.

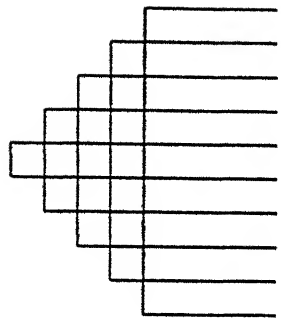


Figure 4.1: Pyramid structure of Empty Tracks

4.1 Two-Terminal Nets

In this section we discuss the routing of two-terminal nets. The nets are classified into *falling*, *rising*, and *vertical* nets. A net is said to be a *falling* net, if the leftmost terminal is on the top of the channel and rightmost terminal is on the bottom of the channel. Similarly a net is said to be a *rising* net, if the leftmost terminal is on the bottom of the channel and rightmost terminal is on the top of the channel. A net is said to be a *vertical* net, if both the terminals are in the same column.

To route the two-terminal nets, $2d-1$ tracks will be required. Odd numbered tracks are used for actual routing, and even numbered tracks are used for layer changes. The tracks are numbered such that *Track 1* is the bottom most track, and *Track 2d-1* is the top most track. Whenever a net changes from one layer to another a *contact cut* will be used.

4.1.1 Serial Algorithm

The algorithm routes the nets such that, at any point in time the empty tracks are in the middle of the channel. Whenever a falling net i ends, falling net in the highest track will occupy the track vacated by net i. Similarly whenever a rising net i ends, rising net in the lowest available track will occupy the track vacated by net i. A *pyramid* structure of empty tracks is maintained in the middle of the channel as shown in Fig 4.1. This structure guarantees that whenever a net backtracks, it would not collide with any other nets.

The algorithm proceeds column by column from left to right of the channel. A net i is said to be *continuing* in column c, if the leftmost terminal of net i is to the left of c, and rightmost terminal is to the right of c. The algorithm is as follows:

1. If a falling net ends, then route it vertically downwards till the bottom of the channel is reached(see Fig 4.2(a)). Similarly if a rising net ends, then route it vertically upwards till the top of the channel is reached.

2. Falling net in the highest track will occupy the track vacated by the falling ending net(see Fig 4.2(b)). Similarly rising net in the lowest track will occupy the track vacated by the rising ending net.

3. If a falling net starts, route it directly into the lowest available track(see Fig 4.2(c)).

4. If no falling net starts, then route the starting rising net directly into the highest available track(see Fig 4.3(a)); otherwise the net is backtracked before being routed into highest available track(see Fig 4.3(b)).

5. Route the vertical net directly(see Fig 4.3(c)).

The serial algorithm takes $O(ld)$ time, where l is length of the channel and d is density of the channel.

4.1.2 Parallel Algorithm

Now we will describe the $O(\log l)$ time optimal parallel algorithm for routing two-terminal nets, on CREW model.

1 For each net find the initial track in which the net should be routed.

Remark: A falling net will always be routed in the lowest available track. Similarly a rising net will always be routed in the highest available track. A net i is a falling net, if $TopTerminal(i) < BottomTerminal(i)$. Similarly a net i is a rising net, if $TopTerminal(i) > BottomTerminal(i)$. Two auxiliary arrays $B[1..2l]$ and $B'[1..2l]$ are used. Initialize the arrays to zero. If net i is a falling net and if $TopTerminal(i) = j$, and $BottomTerminal(i) = k$, then put $B[2j] = 1$, and

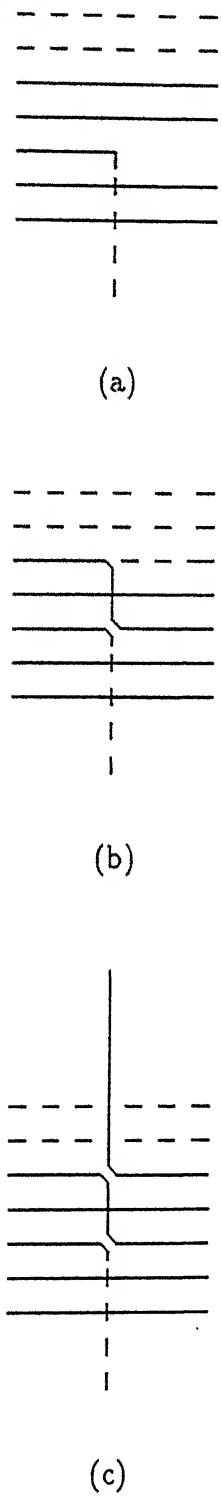
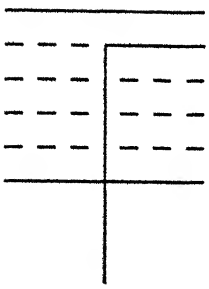
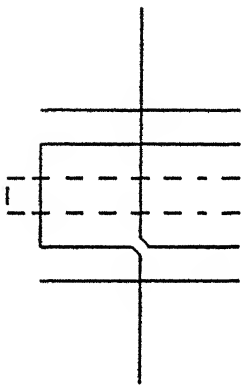


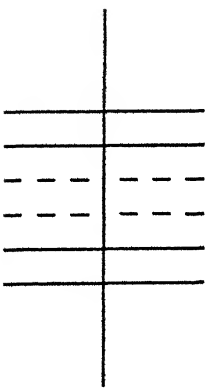
Figure 4.2: Routing of two-terminal nets. (a)Routing of an ending net. (b)Jogging of continuing nets. (c) Routing of a falling net



(a)



(b)



(c)

Figure 4.3: Routing of two-terminal nets. (a) Direct routing of a rising net. (b) Backtracking of a rising net. (c) Routing of a vertical net

$B[2k-1] = -1$; else if the net is a rising net and if $BottomTerminal(i) = j$, and $TopTerminal(i) = k$, then put $B'[2j] = 1$, and $B'[2k-1] = -1$. Find prefix-sums for B and B'. $B[2j]$ gives the initial track for net i, if i is a falling net. Similarly $B'[2j]$ gives the initial track for net i, if i is a rising net. This step can be done in $O(\log l)$ time with $\frac{l}{\log l}$ processors on EREW model, hence also on CREW model.

2 For each net find the track in which it will be there at the time of ending.

2(a) Let s_i, and e_i be the starting and ending columns for a net i. Construct a graph G, in which the nodes represent the falling nets. Put an edge from i to j, if the number of falling nets leaving s_i (a net is said to be leaving the column k if it crosses the channel from k^{th} column to $(k+1)^{th}$ column) is less than number of falling nets leaving e_j(excluding the falling net starting in column e_j), and e_j is the first such column after s_i. Similarly construct a graph G', in which the nodes represent the rising nets. Put at edge from i to j, if the number of rising nets leaving s_i is less than number of rising nets leaving e_j(excluding the rising net starting in the column e_j), and e_j is first such column after s_i. The graphs G and G' contains cycles only(see Claim 4.3 of Section 4.1.3).

Remark: Find the nearest smaller on the right for each item in the vectors B, and B'. Let i be a falling net, and nearest smaller on the right for $B[2s_i]$ be $B[2e_j-1]$. Then put an edge from i to j in the graph G. Similarly let i be a rising net, and nearest smaller on the right for $B'[2s_i]$ be $B'[2e_j-1]$. Then put an edge from i to j in the graph G'. Nearest smaller on the right for each item in a vector of length $2l$ can be found in $O(\log l)$ with $\frac{l}{\log l}$ processors.

2(b) For each cycle in graphs G, and G', find the net i having maximum ending column. Delete the edge to i in the cycle(since there will be no net to occupy its track when it ends). After deleting the edges, we have a set of linked lists.

Remark: Basically we have to find the node having maximum ending column in each cycle. This step can be done in $O(\log l)$ time with $\frac{l}{\log l}$ processors on EREW model.

2(c) List rank the linked lists.

Remark: This step can be done in $O(\log l)$ time with $\frac{l}{\log l}$ processors on EREW model.

2(d) Consider the ending column for each net in the lists obtained in step 2(b). Find the nearest larger on the right for each net in its list. If the nearest larger on the right for net i is net j, then the initial track of net before j in the list, will give the track, in which i will be there, when i ends(see Claim 4.5 of Section 4.1.3).

Remark: We assume that there is an imaginary net at the end of each list, having the ending column as the maximum of all the ending columns in the list, there by every net will have a nearest larger on the right in its list. Nearest larger on the right for each item in a vector of length l can be found in $O(\log l)$ time with $\frac{l}{\log l}$ processors.

3 Find the number of tracks required to realize the layout.

Remark: Two auxiliary arrays $D[1..l]$, and $D'[1..l]$ are used. Let $D[c] = B[2c]$, and $D'[c] = B'[2c]$, $1 \leq c \leq l$. $D[c]$ gives the number of falling nets leaving the column c. Similarly $D'[c]$ gives the number of rising nets leaving the column c. The number of tracks required is equal to $\max(D[c] + D'[c])$, $1 \leq c \leq l$. Maximum of l numbers can be found in $O(\log l)$ time with $\frac{l}{\log l}$ processors.

4 For each rising net find the column till which the net should be backtracked.

Remark: Let i be the rising net starting in the column c. Let a and b be the lowest and highest available tracks at c. The net i can be backtracked till the column j in the track a, if the number of falling nets leaving column j is $a-1$, and the number of falling nets leaving column $j-1$ is a. If there is no such column, the net i can be backtracked till $(-a+1)^{th}$ column, in the track a. Similarly, the net i can be backtracked till the column k in the track b, if the number of rising nets leaving column k is $b-1$, and the number of rising nets leaving column $k-1$ is b. If there is no such column, the net i can be backtracked till $(-b+1)^{th}$ column, in the track b. Let x be the column till which the net can be back tracked in the track a, and x' be the column till which the net i can be backtracked in the

track b, of the channel. $\max(x, x')$ will give the actual column till which the net i should be backtracked.

For each item i in the arrays D and D', find the index of the nearest larger on the right. Let the nearest larger on the right for $D[x]$ be $D[c]$, and $D[x-1] = D[c]$. Similarly let the nearest larger on the right for $D'[x']$ be $D'[c]$, and $D'[x'-1] = D'[c]$. If there is no item on the left for $D[c]$, such that $D[c]$ is the nearest larger on the right for $D[x]$, and $D[x-1] = D[c]$, then $x = -D[c] + 1$. If there is no item on the left for $D'[c]$, such that $D'[c]$ is the nearest larger on the right for $D'[x']$, and $D'[x'-1] = D'[c]$, then $x' = -D'[c] + 1$. Then the rising net starting in the column c can be backtracked till the column $\max(x, x')$. Given l items, *nearest larger* on the right for each item can be found in $O(\log l)$ time with $\frac{l}{\log l}$ processors on CREW model.

5 Find the layer assignments for horizontal segments.

Remark: A two dimensional array $E[1..d][0..l-1]$ is used. Initialize $E[1..d][0..l-1]$ to zero. If a falling net starts at c^{th} column in t^{th} track, then put $E[t][c] = 1$. Similarly if a rising net, that do not need backtracking starts at c^{th} column in t^{th} track, then put $E[t][c] = 1$. Let i be a rising net, that need backtracking, and c be the column till which the net can be backtracked. Let a and b be the highest and lowest available tracks. Put $E[a][c] = 1$, $E[b][c] = 1$, if $c > 0$; else put $E[a][0] = 1$, $E[b][0] = 1$. Find prefix-sums for each row of E. Let a segment starts in j^{th} track and k^{th} column. If the kth prefix-sum of $E[j]$ is odd, then the segment is wired in *layer 1*; else the segment is wired in *layer 2*. This step can be done in $O(\log l)$ time with $\frac{ld}{\log l}$ processors.

6 Find the layer assignments for vertical segments.

Remark: If the horizontal wire at (i, j) is routed in *layer 1*, then vertical wire at $(i, j-1)$ will be routed in *layer 2*, and vice-versa. This step can be done in $O(1)$ time with ld processors, hence also in $O(\log l)$ time with $\frac{ld}{\log l}$ processors on EREW model.

7 Realize the layout.

Remark: We know the layer assignments for horizontal and vertical segments. Route the segments according to the layer assignments.

All steps in the algorithm can be done in $O(\log l)$ time. The maximum number of processors used at any step is at most $\frac{ld}{\log l}$. Hence the following theorem:

Theorem 4.1 *Channel routing problem of two-terminal nets in Knock-Knee model can be solved optimally in $O(\log l)$ time with $\frac{ld}{\log l}$ processors on CREW model, using $2d-1$ tracks.* ∎

4.1.3 Proof of Correctness

The main step we have to prove in the algorithm is Step 2. We prove this step for falling nets. The proof also holds for rising nets. Whenever we refer a net i, we mean falling net i.

Let s_i be the column in which net i starts, and e_i be the column in which net i ends. Put a "1" in $B[2s_i]$, and a "-1" in $B[2e_i-1]$. Compute the prefix-sums on B. $B[2c-1]$ gives the number of nets crossing the column c (excluding the net starting in column c), and $B[2c]$ gives the number of nets crossing the column c. The nets are routed such that at any point in time the nets are packed to one side of the channel. Since a net i starting in column s_i is always routed in the lowest available track, $B[2s_i]$ gives the track in which the net i should be routed when it starts.

Claim 4.1 *A net i routed in track t at column c, will jog at column c', if the number of nets leaving the column c'(excluding net starting in column c) is $t-1$, and the net i crosses the column c', and c' is the first such column after c.*

Proof: The nets are routed such that at any point in time all the nets are packed to one side of the channel. If there are k nets at column j, k tracks are sufficient to route the nets. The number of nets entering the column c' is t. $t-1$ nets are leaving column c(excluding net starting in column c'). In order to pack the nets to one side of the channel, the net in the highest track(i.e the net in track t, which is net i) should occupy the track vacated by the net ending in column c'. ∎

Claim 4.2 *A net i routed in track t at column c, will not jog after column c, if net i ends before the column c', where c' is the first column after c, such that the number of nets leaving column c'(excluding the net starting in the column c') is $t-1$.* ∎

Construct a graph G such that the nodes represent the nets. Put an edge from i to j, if the nearest smaller on the right for $B[2s_i]$ is $B[2e_j - 1]$.

Claim 4.3 *The graph G contains cycles only.*

Proof: We prove this claim by showing that,

1. The out degree of each node is 1, and
2. The in degree of each node is 1.

The graph with these properties contains cycles only.

The array B contains only -1,0, and 1. If there are n nets, obviously there will be n "1's" and n "-1's" in the array B. Since the sum of n "1's" and n "-1's" is zero, the last item $B[2l]$ will be zero. $B[2c-1]$, and $B[2c]$, $1 \leq c \leq l$ gives the number of nets leaving column c(excluding the net starting in column c), and number of nets leaving column c respectively. Hence $B[2c-1]$, and $B[2c]$ will never be negative. Since $B[2s_i] = B[2s_i - 1] + 1$, and $B[2s_i - 1]$ is non-negative, $B[2s_i]$ must be at least 1. The prefix-sum decreases at k^{th} location if a net ends in column $\frac{k+1}{2}$. Since each $B[2s_i]^{th}$ item will have a nearest smaller on the right, the out degree of each node must be 1.

We now show that the in degree of each node in the graph is 1. Let us assume that there is an edge to k from both i, and j, and $s_i < s_j$. The vector obtained in step contains only $-1, 0$ or 1. The prefix-sums in array B, if decreases, it should be in steps of 1 only. Hence, the nearest smaller on the right for an item a should be $a-1$. Since no net will end in column 1, $B[1]$ will always be zero. Since k has edges from both i, and j, $B[2e_k - 1]$ is the nearest smaller on the right to both $B[2s_i]$ and $B[2s_j]$. Hence $B[2s_i]$ should be equal to $B[2s_j]$, and all the items between $B[2s_i]$ and $B[2s_j]$ should not be smaller than $B[2s_i]$. Since $B[2s_j] = B[2s_j - 1] + 1$, $B[2s_j - 1]$ is smaller than $B[2s_j]$, and $B[2s_j - 1]$ is in between $B[2s_i]$ and $B[2s_j]$. Which is a contradiction. Hence the in degree of each node is at most 1. Since the out degree of each net is 1, and the in degree of each net can not be more than 1, the in degree of each net must be 1.

Hence the graph contains cycles only. ∎

Let i be the net routed in track t at column c. The meaning of the edge from i to j is that, the number of nets leaving c_j(excluding the net starting in column e_j) is $t-1$, and c_j is the first such column after c.

If there is an edge from i to j, and i ends after j ends, net i will occupy the track vacated by net j(see Claim 4.1). If there is an edge from i to j, and i ends before j ends, then i will never jog(see Claim 4.2), and it will be in its initial track when it ends.

The graph may contain more than one cycle. The nets in one cycle will never interact directly with the nets in other cycles. When a net having a maximum column number in its cycle ends, there will be no net in the cycle to occupy the track vacated by it (since all nets ends before it). Delete the edge to the net having maximum ending column. This will result in a set of lists.

Proceed column by column from left to right of the channel. Let j be the net ending in the current column. Let i and k be the predecessor and successor nets for net j in the list. Set the track of net i as track of net j, and set the successor of net i as k.

Let us assume that the ending columns of all the nets between net i and net k in the list are less than the ending columns of net i and net k.

Claim 4.4 *When all the nets between net i and net k ends, there will be an edge from net i to net k.*

Proof: We prove this claim by induction on the number of nets between i and k.

When there are 0 nets between net i and net k, obviously there will be en edge between net i and net k. Thus the claim holds when there are 0 nets between i and k.

When there is only one net(say a) between net i and net k. Since a ends before i and k ends, a will set the successor of net net i as successor of net a. The successor of a is k. Hence when all the nets between i and k ends, there will be an edge from i to k. Thus the claim holds when there is 1 net between i and k.

Let us assume that the claim holds when there are n nets between i and k. We prove that the claim also holds when there are $n+1$ nets between i and k.

Let j be the net having maximum ending column between net i and net k. The number of nets between net i and net j will be at most n, and the number of nets

between net j and net k will be at most n. By induction hypothesis, at the time when net b ends, there will be edges from i to j, and j to k. When j also ends, j will set the successor of net i as successor of net j. The successor of j is k. Hence, when all the nets between i and k ends, there will be an edge form i to k.

Thus the claim holds for any number of nets between net i and net k. ∎

Claim 4.5 *When net i ends, it will be in the track of net j, where j is the predecessor of net k, and k is the nearest larger on the right for net i.*

Proof: Since k is the nearest larger on the right for i, ending column of all the nets between net i and k is less than the ending column of net i. Hence all the nets between i and k will end before net i ends.

We prove this claim by induction on the number of nets between net i and net k.

When there are 0 nets between net i and net k, the predecessor of net k is i. Hence when i ends it will be in the track of net i(see Claim 4.2).

When there is only one net(say a) between net i and net k. Since a ends before net i ends, a will set the track of net i as track of net a. There will be an edge from i to k(see Claim 4.4). Hence when i ends it will be in the track of net a(see Claim 4.2), and a is the predecessor of net k. Hence the claim holds.

Let us assume that the claim holds when there are n nets between i and k. We prove that the claim also holds when there are $n+1$ nets between i and k.

Let b be the net having maximum ending column between net i and net k. The nearest larger on the right for net b is obviously k. Let a be the predecessor of net b, and j be the predecessor of net k. The number of nets between net i and net b will be at most n, and the number of nets between net b and net k will be at most n. By induction hypothesis, at the time when net b ends, net b will be in the track of net j. There will be edges from i to b, and b to k(see Claim 4.4). When b ends, it will set the track of net i as track of net b, which is track of net j. There will be an edge from i to k. Hence when i ends it will be in the track of net j(see Claim 4.2), which is the predecessor of net k. Hence the claim holds for any number of nets between net i and net k. ∎

Thus Step 2 of the algorithm works correctly.

4.1.4 Example

In this section we will give an example for the algorithm discussed in the previous section. Let the given nets be

Net	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
TT	1	2	3	4	6	8	9	10	11	12	13	14	15	16	17	18	19	20
BT	10	7	8	14	13	3	15	2	16	20	1	9	12	18	19	21	11	5

(TT = $TopTerminal$; BT = $BottomTerminal$)

Here nets $1, 2, 3, 4, 5, 7, 9, 10, 14, 15, 16$ are falling nets, and nets $6, 8, 11, 12, 13, 17, 18$ are rising nets.

1 For each net find the initial track in which the net should be routed.

$Column$	1	2	3	4	5	6	7	8	9	10	11	12	13	14
B	0	1	0	1	0	1	0	1	0	0	0	1	−1	0
B'	0	1	0	1	0	1	0	0	0	1	0	0	0	0

$Column$	15	16	17	18	19	20	21	22	23	24	25	26	27	28
B	−1	0	0	1	−1	0	0	1	0	1	−1	0	−1	0
B'	−1	0	0	1	−1	0	0	1	0	1	−1	0	−1	0

$Column$	29	30	31	32	33	34	35	36	37	38	39	40	41	42
B	−1	0	−1	1	0	1	−1	1	−1	0	−1	0	−1	0
B'	−1	0	0	0	0	0	0	0	−1	0	−1	0	0	0

The prefix-sums of B and B' are

$Column$	1	2	3	4	5	6	7	8	9	10	11	12	13	14
B	0	1	1	2	2	3	3	4	4	4	4	5	4	4
B'	0	1	1	2	2	3	3	3	3	4	4	4	4	4

Column	15	16	17	18	19	20	21	22	23	24	25	26	27	28
B	3	3	3	4	3	3	3	4	4	5	4	4	3	3
B'	3	3	3	4	3	3	3	4	4	5	4	4	3	3

Column	29	30	31	32	33	34	35	36	37	38	39	40	41	42
B	2	2	1	2	2	3	2	3	2	2	1	1	0	0
B'	2	2	2	2	2	2	2	2	1	1	0	0	0	0

The initial tracks for the falling nets(from the channel bottom) are

Net	1	2	3	4	5	7	9	10	14	15	16
Track	1	2	3	4	5	4	4	5	2	3	3

The initial tracks for the rising nets(from the channel top) are

Net	11	8	6	18	12	17	13
Track	1	2	3	4	4	4	5

2 For each net find the track in which the net will be there at the time of ending.

2(a) Construct graphs G and G'(see Fig 4.4(a),4.4(b)).

2(b) For each cycle in graphs G, and G', find the net i having maximum ending column. Delete the edge to i in the cycle. After deleting the edges, we have a set of linked lists(see Fig 4.4(c),4.4(d)).

2(c) List rank the linked lists.

2(d) Consider the ending column for each net in the lists obtained. Find the nearest larger on the right for each net in its list. If the nearest larger on the right for net

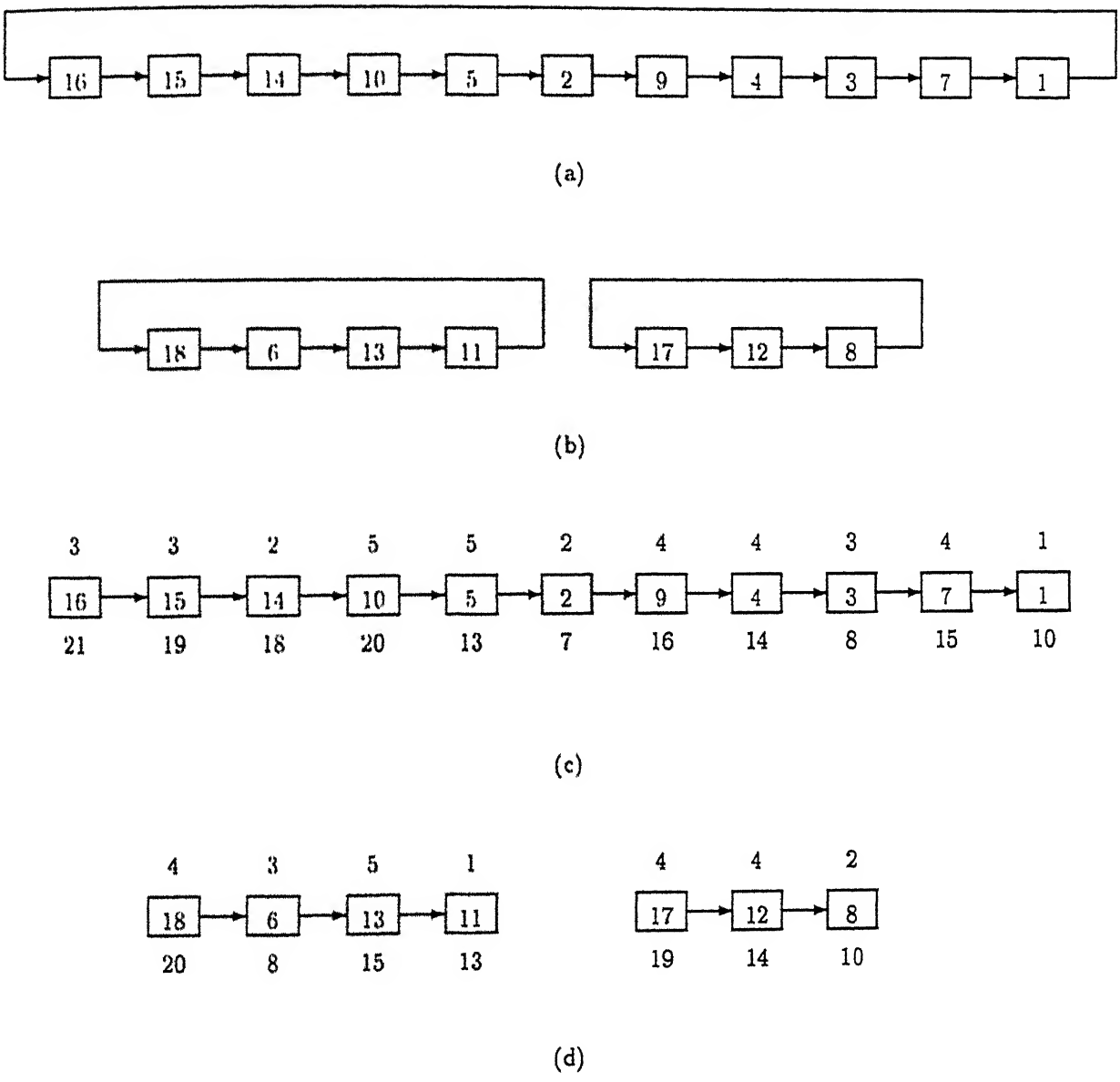


Figure 4.4: (a) The graph G. (b) The graph G'. (c) The graph G, after deleting the edge to the net having maximum ending column in each cycle. (d) The graph G' after deleting edge to the net having maximum ending column in each cycle. The number above and below the nets in (c) and (d) represent the initial track of the net, and the ending column of the net.

i is net j, then the initial track of net before j in the list, will give the track, in which i will be there, when i ends.

The track for the falling net when the net ends is

Net	16	15	14	10	5	2	9	4	3	7	1
Track	1	2	2	1	2	2	1	3	3	1	1

The track for the rising nets when the net ends is

Net	18	6	13	11	17	12	8
Track	1	3	1	1	2	2	2

3 Find the number of tracks required to realize the layout.

D	1	2	3	4	4	5	4	3	4	3	4	5	4	3	2	2	3	3	2	1	0
D'	1	2	3	3	4	4	4	3	4	3	4	5	4	3	2	2	2	2	1	0	0
$D + D'$	2	4	6	7	8	9	8	6	8	6	8	10	8	6	4	4	5	5	3	1	0

Maximum is 10. Hence 10 tracks are required to realize the layout.

4 For each rising net find the column till which the net should be backtracked.

Rising net 11 backtracks till 0th column.
Rising net 8 backtracks till -1st column.
Rising net 6 backtracks till -2nd column.
Rising net 12 backtracks till 8th column.
Rising net 17 backtracks till 10th column.
Rising net 13 backtracks till 7th column.

5 Find the layer assignments for horizontal segments.

The Matrix E is

$Track10$	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1
$Track9$	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1	0
$Track8$	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
$Track7$	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
$Track6$	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
$Track5$	0	0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0
$Track4$	0	0	0	0	1	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0
$Track3$	1	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1	0	0
$Track2$	1	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0	1	1	0
$Track1$	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	1

Find the prefix-sums for each row of E. The segment starting in t^{th} track at c^{th} column is routed in *layer 1* if $E[t][c]$ is odd; otherwise the segment is routed in *layer 2*.

6 Find the layer assignments for vertical segments.

7 Realize the layout(see Fig 4.5, layer changes are not shown).

4.1.5 Modified Serial Algorithm

This algorithm differs from the original algorithm in Step2. In the algorithm of Berger et. al[3], rising net in the lowest track will occupy the track vacated by a rising net. Similarly, falling net in the highest track will occupy the track vacated by a falling net. In the modified algorithm, when a rising net ends, all the rising nets below the ending rising net will move one track up. Similarly when a falling net ends, all the falling nets above the falling ending net will move one track down.

The algorithm proceeds column by column from left to right of the channel. The algorithm is as follows:

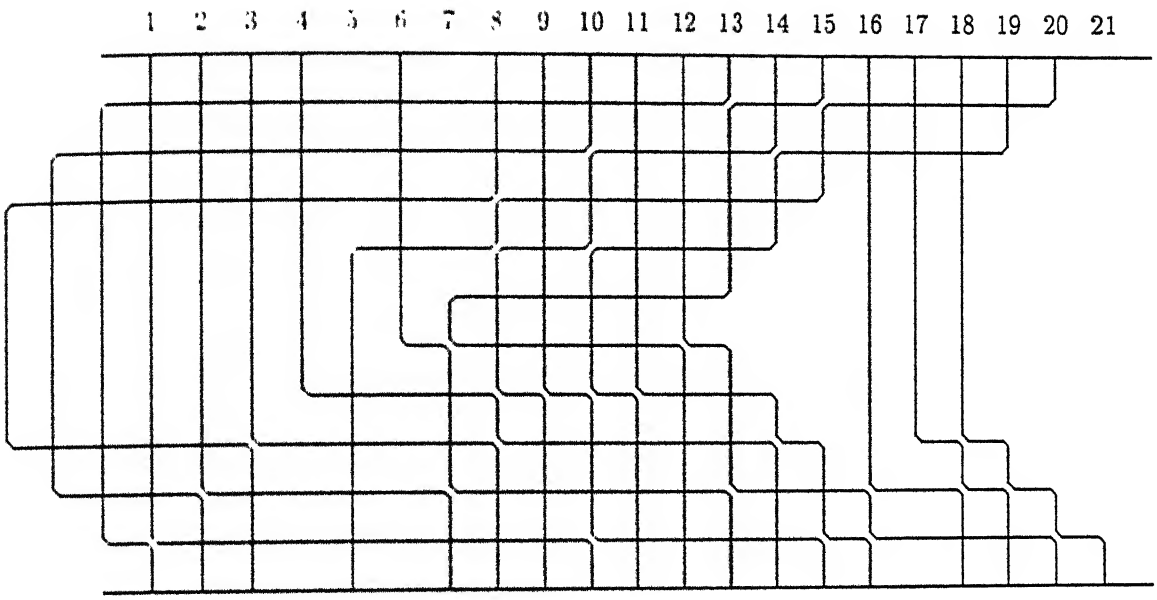


Figure 4.5: The layout for the given example

1. If a falling net ends, then route it vertically downwards till the bottom of the channel is reached(see Fig 4.6(a)). Similarly, if a rising net ends, then route it vertically upwards till the top of the channel is reached.

2. Fill the track vacated in Step 1 by jogging the continuing nets (see Fig 4.6(b)).

3. If a falling net starts, route it directly into the lowest available track (see Fig 4.6(c)).

4. If no falling net starts , then route the starting rising net directly into the highest available track (see Fig 4.7(a)); otherwise the net is backtracked before being routed into highest available track(see Fig 4.7(b)).

5. Route the vertical net directly(see Fig 4.7(c)).

The algorithm takes $O(ld)$ time, where l is length of the channel and d is density of the channel.

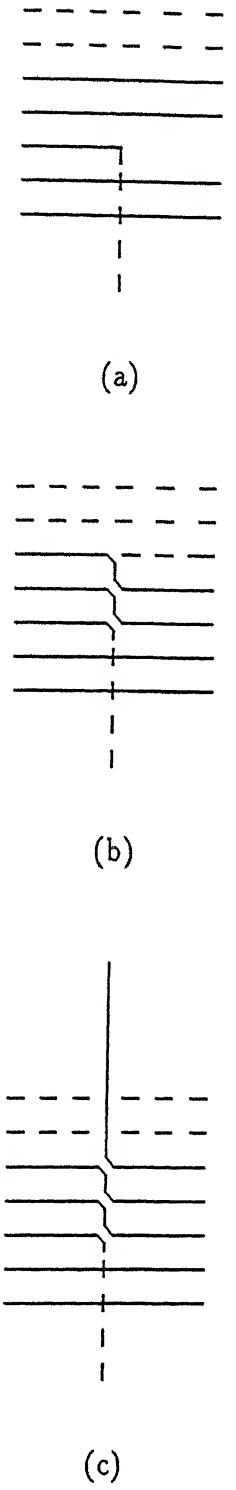
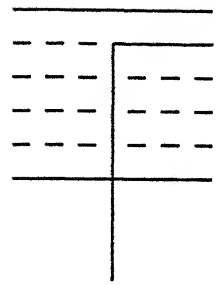
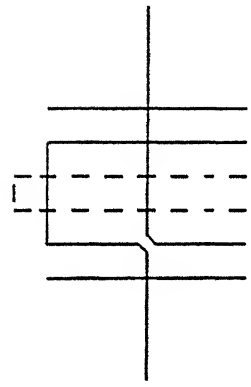


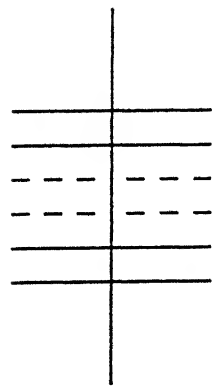
Figure 4.6: Routing of two-terminal nets. (a)Routing of an ending net. (b)Jogging of continuing nets. (c) Routing of a falling net



(a)



(b)



(c)

Figure 4.7: Routing of two-terminal nets. (a) Direct routing of a rising net. (b) Backtracking of a rising net. (c) Routing of a vertical net

4.1.6 Parallel Algorithm

Now we will describe the optimal $O(\log l)$ time parallel algorithm for routing two-terminal nets, on CREW model. The parallel algorithm is

1. Divide the nets into falling, rising and vertical nets. Sort the falling nets based on their position of top terminals, and rising nets based on their position of bottom terminals.

   Remark: A net i is a falling net, if $TopTerminal(i) < BottomTerminal(i)$. Similarly, a net i is a rising net, if $BottomTerminal(i) < TopTerminal(i)$. Two auxiliary arrays $A[1..l]$ and $A'[1..l]$ are used. Initialize the arrays to zero. If net i is a falling net and $TopTerminal(i) = k$, then $A[k] = 1$. If the net i is a rising net and $BottomTerminal(i) = k$, then $A'[k] = 1$. Find prefix-sums for A and A'. $A[k]$ gives the index of net i in the sorted sequence of falling nets, if net i is a falling net. $A'[k]$ gives the index of net i in the sorted sequence of rising nets, if net i is a rising net. Prefix-sums of l numbers can be found in $O(\log l)$ time with $\frac{l}{\log l}$ processors on EREW model, hence also on CREW model.

2. For each net find the initial track in which the net should be routed.

   Remark: A falling net will always be routed in the lowest available track. Similarly, a rising net will always be routed in the highest available track. Two auxiliary arrays $B[1..l]$ and $B'[1..l]$ are used. Initialize the arrays to zero. If net i is a falling net and $BottomTerminal(i) = k$, then $B[k] = -1$; else if the net is a rising net and $TopTerminal(i) = k$, then $B'[k] = -1$. Find prefix-sums for B and B'. $A[k] + B[k]$ gives the initial track(from channel bottom) for net i, if i is a falling net and $TopTerminal(i) = k$. Similarly, $A'[k] + B'[k]$ gives the initial track(from channel top) for net i, if i is a rising net and $BottomTerminal(i) = k$. Prefix-sums of l numbers can be found in $O(\log l)$ time with $\frac{l}{\log l}$ processors on EREW model, hence also on CREW model.

3. For each net find the columns at which the net should jog.

   Remark: A falling net i will jog in column k, if a falling net(say j) ends in column k, and net i starts after j starts, and net i is crossing the column k. Similarly, a

rising net i will jog in column k, if a rising net j ends in column k, and net i starts after j starts, and net i is crossing the column k.

Two auxiliary arrays $S[1..l]$, and $S'[1..l]$ are used. Initialize S, and S' to zero. If net i is a falling net, and $BottomTerminal(i) = k$, then $S[k] = 1$. Similarly, if net i is a rising net, and $TopTerminal(i) = k$, then $S'[k] = 1$.

Let i be a falling net and $x = TopTerminal(i)$ and $y = BottomTerminal(i)$. Let $P_i = y - x$. Assign $\frac{P_i}{\log l}$ processors to each falling net i; we also use a local array $C[i][x+1..y]$; $C[i][k] = 1$, if $S[k] = 1$, $BottomTerminal(j) = k$, and $j < i$; otherwise $C[i][k] = 0$; $x < k \leq y$. The index of 1's in the array $C[i]$ will give the columns at which the falling net i should jog. Find the prefix-sums for $C[i]$. Let t be the initial track for net i, then $t - C[i][k]$ will give the track number at column k, from the channel bottom.

Let i be a rising net and $y = TopTerminal(i)$ and $x = BottomTerminal(i)$. Let $P'_i = y - x$. Assign $\frac{P'_i}{\log l}$ processors to each falling net i; we also use a local array $C'[i][x+1..y]$; $C'[i][k] = 1$, if $S'[k] = 1$, $BottomTerminal(j) = k$, and $j < i$; otherwise $C'[i][k] = 0$; $x < k \leq y$. The index of 1's in the array $C'[i]$ will give the columns at which the rising net i should jog. Find the prefix-sums for $C'[i]$. Let t be the initial track for net i, then $t - C'[i][k]$ will give the track number at column k, from the channel top.

Each net i will do this step in $O(\log l)$ time with $\frac{P_i}{\log l}$ processors if i is a falling net or with $\frac{P'_i}{\log l}$ processors if i is a rising net, on CREW model. As $\sum P_i + \sum P'_i \leq ld$, this step can be done in $O(\log l)$ time, with $\frac{ld}{\log l}$ processors.

4. Find the number of tracks required to realize the layout.

   Remark: Add vectors A and B to give vector D. Similarly, add vectors A' and B' to give vector D'. $D[i]$ gives the number of falling nets leaving the column i (a net is said to be *leaving* the column, if it crosses the channel from i^{th} column to $(i+1)^{th}$ column). Similarly, $D'[i]$ gives the number of rising nets leaving the column i. The number of tracks required is equal to $\max(D[i] + D'[i])$, $1 \leq i \leq l$. Maximum of l numbers can be found in $O(\log l)$ time with $\frac{l}{\log l}$ processors.

5. For each rising net find the column till which the net should be backtracked.

   Remark: Let i be the rising net starting in the column c. Let a and b be the lowest and highest available tracks at c. The net i can be backtracked till the column j in the track a, if the number of falling nets leaving column j is $a-1$, and the number of falling nets leaving column $j-1$ is a. If there is no such column, the net i can be backtracked till $(-a+1)^{th}$ column, in the track a. Similarly, the net i can be backtracked till the column k in the track b, if the number of rising nets leaving column k is $b-1$, and the number of rising nets leaving column $k-1$ is b. If there is no such column, the net i can be backtracked till $(-b+1)^{th}$ column, in the track b. Let x be the column till which the net can be backtracked in the track a, and x' be the column till which the net i can be backtracked in the track b. $\max(x, x')$ will give the actual column till which the net i should be backtracked.

   For each item i in the arrays D and D', find the index of the nearest larger on the right. Let the nearest larger on the right for $D[x]$ be $D[c]$, and $D[x-1] = D[c]$. Similarly let the nearest larger on the right for $D'[x']$ be $D'[c]$, and $D'[x'-1] = D'[c]$. If there is no item on the left for $D[c]$, such that $D[c]$ is the nearest larger on the right for $D[x]$, and $D[x-1] = D[c]$, then $x = -D[c]+1$. If there is no item on the left for $D'[c]$, such that $D'[c]$ is the nearest larger on the right for $D'[x']$, and $D'[x'-1] = D'[c]$, then $x' = -D'[c]+1$. Then the rising net starting in the column c can be backtracked till the column $\max(x, x')$. Given l items, *nearest larger* on the right for each item can be found in $O(\log l)$ time with $\frac{l}{\log l}$ processors on CREW model.

6. For each net find the layer assignments for its horizontal segments.

   Remark: A two dimensional array $E[0..d][1..l]$ is used. Initialize $E[0..d][1..l]$ to zero. For each net we know the columns at which the net should jog. If a net i jogs to the track t in column c, then put $E[t][c] = 1$(i.e a horizontal segment is starting from $E[t][c]$). If a falling net starts at c^{th} column in t^{th} track, then put $E[t][c] = 1$. Similarly, if a rising net, that do not need backtracking starts at c^{th} column in t^{th} track, then put $E[t][c] = 1$. Let i be a rising net that need backtracking, and c be the the column till which the net can be backtracked. Let

a and b be the highest and lowest available tracks. Put $E[a][c] = 1$, $E[b][c] = 1$, if $c > 0$; else put $E[a][0] = 1$, $E[b][0] = 1$. Find prefix-sums for each row of E. Let a segment starts in j^{th} track and k^{th} column. If the kth prefix-sum of $E[j]$ is odd then the segment is wired in *layer 1*, else the segment is wired in *layer 2*. This step can be done in $O(\log l)$ time with $\frac{ld}{\log l}$ processors.

7. For each net find the layer assignments for its vertical segments.

   Remark: If the horizontal wire at (i, j) is routed in *layer 1*, then vertical wire at $(i, j-1)$ will be routed in *layer 2*, and vice-versa. This step can be done in $O(1)$ time with ld processors, hence also in $O(\log l)$ time with $\frac{ld}{\log l}$ processors on CREW model.

8. Realize the layout.

   Remark: We know the layer assignments for horizontal and vertical segments of each net. Route the net according to the layer assignments.

All steps in the algorithm can be done in $O(\log l)$ time. The maximum number of processors used at any step is at most $\frac{ld}{\log l}$. Hence the following theorem:

Theorem 4.2 *Channel Routing problem of two-terminal nets in Knock-Knee model can be solved optimally in $O(\log l)$ time with $\frac{ld}{\log l}$ processors on CREW model, using $2d - 1$ tracks.* ∎

4.1.7 Example

Now we will give an example for the algorithm discussed in the previous section.
Let the given nets be

Net	1	2	3	4	5	6	7	8	9
TopTerminal	1	2	3	5	7	4	9	10	8
BottomTerminal	6	4	8	10	9	1	2	3	7

1. Classify the nets into falling, rising and vertical nets. Sort the falling nets based on position of their top terminal, and rising nets based on position of their bottom terminal.

   Here nets $1, 2, 3, 4$, and 5 are falling nets, and nets $6, 7, 8$, and 9 are rising nets.

A	1	1	1	0	1	0	1	0	0	0
A'	1	1	1	0	0	0	1	0	0	0

   Prefix-sums of A and A' are

A	1	2	3	3	4	4	5	5	5	5
A'	1	2	3	3	3	3	4	4	4	4

2. Find the initial track for each net.

B	0	0	0	-1	0	-1	0	-1	-1	-1
B'	0	0	0	-1	0	0	0	-1	-1	-1

   Prefix-sums of B, and B' are

B	0	0	0	-1	-1	-2	-2	-3	-4	-5
B'	0	0	0	-1	-1	-1	-1	-2	-3	-4

$A+B$	1	2	3	2	3	2	3	2	1	0
$A'+B'$	1	2	3	2	2	2	3	2	1	0

   The initial tracks for the nets are

$C'[1][2..4] =$	0	0	0							
$C'[2][3..9] =$		0	1	0	0	0	0	0		
$C'[3][4..10] =$			1	0	0	0	0	1	0	
$C'[4][8] =$							0			

The prefix-sums for C' are

Column	1	2	3	4	5	6	7	8	9	10
$C[1][2..4] =$		0	0	0						
$C[2][3..9] =$			0	1	1	1	1	1	1	
$C[3][4..10] =$				1	1	1	1	1	2	2
$C[4][8] =$								0		

4. Find the number of tracks required to realize the layout.

D	1	2	3	2	3	2	3	2	1	0
D'	1	2	3	2	2	2	3	2	1	0
$D + D'$	2	4	6	4	5	4	6	4	2	0

The maximum is 6, hence the number of tracks required is 6.

5. For each rising net find the number of units on the left till which the net should be backtracked.

Rising net 6 can be backtracked till 0th column.
Rising net 7 can be backtracked till -1st column.
Rising net 8 can be backtracked till -2nd column.
Rising net 9 can be backtracked till 6th column.

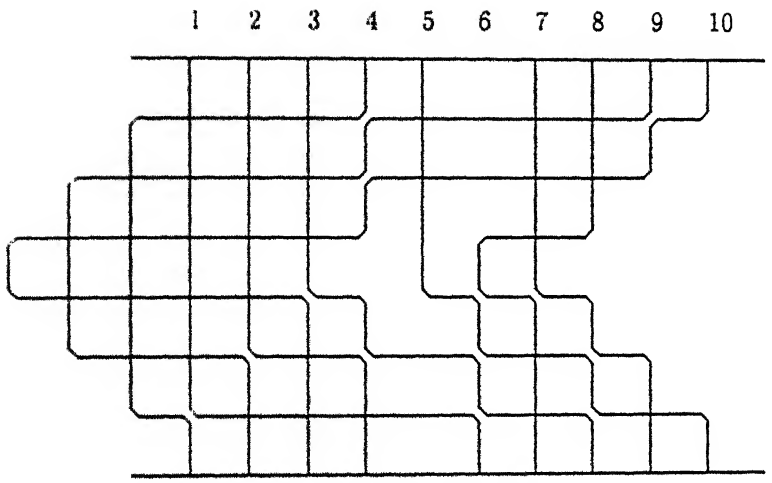


Figure 4.8: The layout for the example

6. Find the layer assignments for horizontal segments.

   The matrix E is

Column	0	1	2	3	4	5	6	7	8	9
Track6	1	0	0	0	1	0	0	0	0	1
Track5	1	0	0	0	1	0	0	0	0	0
Track4	1	0	0	0	0	0	1	0	0	0
Track3	1	0	0	1	0	1	1	1	0	0
Track2	1	0	1	0	1	0	1	0	1	0
Track1	1	1	0	0	0	0	1	0	1	0

   Find the prefix-sums for each row of E. A horizontal segment starting in j^{th} track and k^{th} column is routed in *layer 1* if $E[j][k]$ is odd, otherwise the segment is routed in *layer 2*.

7. Find the layer assignments for vertical segments.

8. Realize the layout(see Fig 4.8, layer changes are not shown).

4.2 Multi-Terminal Nets

In this section we will discuss routing in the presence of multi-terminal nets. The strategy is to create two horizontal wires, one on the rising and one on the falling side of the channel, for each multi-terminal net that has terminals(excluding the leftmost terminal) on both sides of the channel. Each wire is treated as a different net, and will end at its rightmost terminal. This strategy gives rise to several different types of nets. Basically nets are classified into *rising*, *falling* and *vertical* nets. A *falling* net has its leftmost terminal on the top of the channel. A *rising* net has its leftmost terminal on the bottom of the channel. A *vertical* net has both leftmost terminals in the same column. These nets are further classified into *same-side*, *crossing*, and *double-side* nets. In a *same-side* net all the terminals are on one side of the channel. In a *crossing* net the leftmost terminal will be on one side and other terminals will be on other side. In a *double-side* net terminals will be on both sides of the channel(excluding the column in which the net starts).

4.2.1 Serial Algorithm

This algorithm is a modified version of the algorithm due to Berger et. al [3]. The algorithm in the worst case uses $4d - 1$ tracks to route the given multi-terminal nets. As in the case of two-terminal nets, odd numbered tracks are used for actual routing, and even numbered tracks are used for layer changes. Each double-side net is routed in two tracks, one on each side of the channel. The pyramid structure of empty tracks is maintained in the middle of the channel. In order to avoid collisions between rising and falling nets that begin in the same column, rising nets are *backtracked* to the left before being routed into their target tracks. Fig 4.9 and Fig 4.10 shows the routing in a column that contains both a falling net and a rising net.

The tracks are numbered such that *Track 1* is the bottom most track. At any point in time, the region below the empty tracks is called falling side region, and the region above the empty tracks is called rising side region. The algorithm proceeds column by column from left to right of the channel. The algorithm is as follows:

1. If a net in the falling side region ends, then route it vertically downwards till the

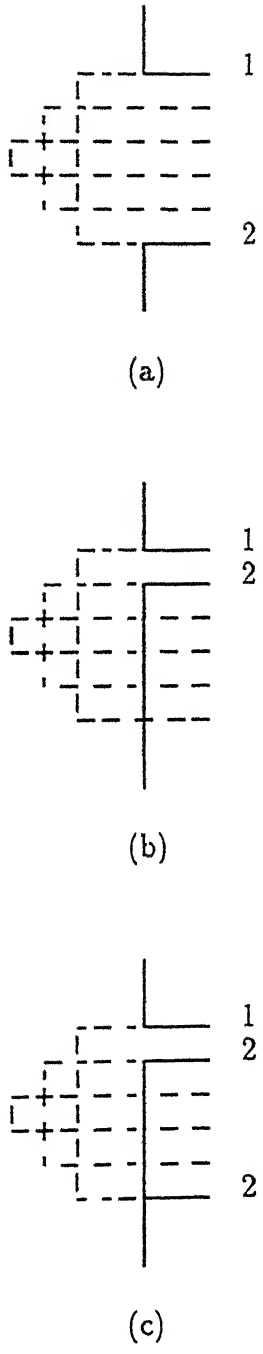


Figure 4.9: Routing of a falling net(1) and a rising net(2) starting in a same column. (a) Two same-side nets. (b) Same-side and crossing nets. (c) Same-side and Double side nets.

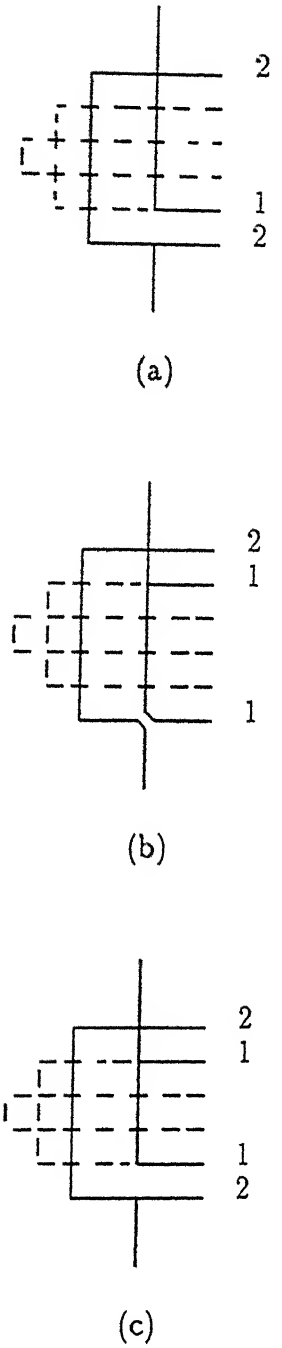


Figure 4.10: Routing of a falling net(1) and a rising net(2) starting in a same column. (a) a falling crossing net and a rising double-side net. (b) a falling double-side net and a rising crossing net (c) Two double-side nets.

channel bottom is reached. Similarly, if a net in the rising side region ends, then route it vertically upwards till the channel top is reached.

2. Fill the track vacated in the falling side region in Step 1, by jogging continuing nets in the falling side region. Similarly, fill the track vacated in the rising side region in Step 1, by jogging continuing nets in the rising side region.

3. If a continuing net in the falling side region has a terminal in the column, then route it vertically downwards till the channel bottom is reached. Similarly, if a continuing net in the rising side region has a terminal in the column, then route it vertically upwards till the channel top is reached.

4. Route same-side falling net in the highest available track. Similarly, route same-side rising net in the lowest available track.

5. If the column does not contain a falling starting net, then route the rising starting net directly into the highest available track, and if the net is a double-side net, also route the net in the lowest available track. If the column contains a falling starting net, then the rising starting net is backtracked before being routed into highest available track, and if the net is a double-side net, also route the net in the lowest available track.

6. If a falling net starts, route it directly into the lowest available track, and if the net is a double-side net, also route the net in the highest available track.

7. Route the vertical net directly. If the net is a vertical rising net then route the net into the highest available track. If the net is a vertical falling net then route the net into lowest available track. If the net is a double-side vertical then route the net in both the highest and lowest available tracks.

Note that in the algorithm of Berger et. al[3], net in the highest track in the falling side region will occupy the vacated track in the falling side region, and net in the lowest track in the rising side region will occupy the track vacated in the rising side region. In the modified algorithm all the nets in the tracks above vacated track will jog to occupy the track vacated by the net in the track below its track, in the falling side region, and

all the nets in the tracks below the vacated track will jog to occupy the track vacated by the net in the track below its track, in the rising side region. The serial algorithm takes $O(ld)$ time, where l is length of the channel and d is density of the channel.

4.2.2 Parallel Algorithm

First a high level description of the algorithm is given. Implementation details will be described later. The high level description of the algorithm is as follows:

1. Sort the nets that require a track in the rising side region, based on the column in which the nets starts. Similarly, sort the nets that require a track in the falling side region, based on the column in which the nets starts.

2. For each net find the initial track in which the net should be routed.

3. For each net find the columns at which the net should jog.

4. For each rising net find the column till which the net can be backtracked.

5. Find the number of tracks required to route the nets.

6. For each net find the layer assignments for its horizontal segments.

7. For each net find the layer assignments for its vertical segments.

8. Realize the layout.

We now describe the implementation details in a more detailed description of the algorithm.

1. Sort the nets that require a track in the rising side region, based on the column in which the nets starts. Similarly, sort the nets that require a track in the falling side region, based on the column in which the nets starts.

   A net will need a track in the rising side region, if the net has a terminal on the top of the channel(excluding the terminal in the column in which it starts). Similarly, a net will need a track in the falling side region, if the net has a terminal

on the bottom of the channel(excluding the terminal in the column in which it starts). There may be more than one net, starting in the same column, that need a track in the rising side region. Similarly, there may be more than one net, starting in the same column, that need a track in the falling side region. The order of preference is same-side or vertical net, then rising net, then falling net. Basically the arrays $T[1..l]$,$T_1[1..l]$, $T_2[1..l]$, $T_3[1..l]$, $B[1..l]$,$B_1[1..l]$, $B_2[1..l]$, $B_3[1..l]$ are used. Initialize all the arrays to zero. If a net that need a track in the rising side region starts in the column c, and if the net is a

same-side or vertical net, then put $T_1[c] = 1$,

rising net, then put $T_2[c] = 1$,

falling net, then put $T_3[c] = 1$.

Let $T[c] = T_1[c] + T_2[c] + T_3[c]$. Find prefix-sums on vector T. The cth prefix-sum of T will give the number of nets started till the cth column, in the rising side region. Then the index of the net starting in the column c in the sorted list of nets that need a track in the rising side region is

$T[c-1] + T_1[c]$ if the net is a same-side or vertical net,

$T[c-1] + T_1[c] + T_2[c]$ if the net is a rising net,

$T[c]$ if the net is a falling net.

Similarly, if a net that need a track in the falling side region starts in the column c, and if the net is a

same-side or vertical net, then put $B_1[c] = 1$,

rising net, then put $B_2[c] = 1$,

falling net, then put $B_3[c] = 1$.

Let $B[c] = B_1[c] + B_2[c] + B_3[c]$. Find prefix-sums on vector B. The cth prefix-sum of B will give the number of nets started till the cth column, in the falling side region. Then the index of the net starting in the column c in the sorted list of nets that need a track in the falling side region is

$B[c-1] + B_1[c]$ if the net is a same-side or vertical net,

$B[c-1] + B_1[c] + T_2[c]$ if the net is a rising net,

$B[c]$ if the net is a falling net.

2. For each net find the initial track in which the net should be routed.

   If a net that require a track in the falling side region ends in column c, then put $E'[c] = -1$. Similarly, if a net that require a track in the rising side region ends in column c, then put $E''[c] = -1$. Find the prefix-sums on vectors E and E'. If the index of the net is i, and starts in column c, and need a track in the falling side region, then $i - E[c]$ will give the initial track(from the channel bottom) for net i. Similarly, if the index of the net is i, and starts in column c, and need a track in the rising side region, then $i - E'[c]$ will give the initial track(from the channel top) for net i.

3. For each net find the columns at which the net should jog.

   Implementation of this step is similar to that of two-terminal nets. A net is said to be a falling side net, if it is routed in the falling side region. Similarly, a net is said to be a rising side net, if the net is routed in the rising side region.

   A falling side net i will jog in column k, if a falling side net j ends in column k, and net i starts after j starts, and net i is crossing the column k. Similarly, a rising side net i will jog in column k, if a rising side net j ends in column k, and net i starts after j starts, and net i is crossing the column k.

   Let s_i be the column in which net i starts, and e_i be the column in which net i ends. Two auxiliary arrays $R[1..l]$, and $R'[1..l]$ are used. Initialize R, and R' to zero. If net i is a falling side net, then $R[e_i] = 1$. Similarly, if net i is a rising side net, then $R'[e_i] = 1$.

   Let i be a falling side net, and $P_i = e_i - s_i$. Assign $\frac{P_i}{\log l}$ processors to each falling side net i; we also use a local array $C[i][x+1..y]$; $C[i][k] = 1$, if $R[k] = 1$, and $k = e_j$ and $j < i$; otherwise $C[i][k] = 0$; $s_i < k \leq e_i$. The index of 1's in the array $C[i]$ will give the columns at which the falling side net i should jog. Find the prefix-sums for $C[i]$, let t be the initial track for net i, then $t - C[i][k]$ will give the track number at column k, from the channel bottom.

Let i be a rising side net and $P'_i = e_i - s_i$. Assign $\frac{P'_i}{\log l}$ processors to each falling net i; we also use a local array $C'[i][x+1..y]$; $C'[i][k] = 1$, if $R'[k] = 1$, $e_j = k$, and $j > i$; otherwise $C'[i][k] = 0$; $s_i < k \leq e_i$. The index of 1's in the array $C'[i]$ will give the columns at which the rising side net i should jog. Find the prefix-sums for $C'[i]$. Let t be the initial track for net i, then $t - C'[i][k]$ will give the track number at column k, from the channel top.

Each net i will do this step in $O(\log l)$ time with $\frac{P_i}{\log l}$ processors, on CREW model. As $\sum P_i + \sum P'_i \leq 2ld$, this step can be done in $O(\log l)$ time with $\frac{ld}{\log l}$ processors, on the CREW model.

4. For each rising net find the column till which the net can be backtracked.

A rising net starting in the column c is backtracked in the track $T[c-1] + T_1[c] + T_2[c] + E'[c]$ in the rising side region, and in the track $B[i-1] + B_1[i] + B_2[i] + E'[i]$ in the falling side region if the rising net is a double-side net; else in the track $B[c] + E[c]$.

Let

$D[c] = B[c-1] + B_1[c] + B_2[c] + E[c]$, if $B_2[i] = 1$

$D[c] = B[c] + E[c]$, otherwise.

Similarly, let

$D'[c] = T[c-1] + T_1[c] + T_2[c] + E'[c]$, if $T_2[i] = 1$

$D'[c] = T[c] + E'[c]$, otherwise.

Let i be the rising net starting in the column c. Let a and b be the lowest and highest available tracks at c. The net i can be backtracked till the column j in the track a, if the number of nets that require a track in the falling side region, leaving column j is $a-1$, and the number of nets that require a track in the falling side region, leaving column $j-1$ is a. If there is no such column, the net i can be backtracked till $(-a+1)^{th}$ column, in the track a. Similarly, the net i can be backtracked till the column k in the track b if the number of nets that require a track in the rising side region, leaving column k is $b-1$, and the number of nets that require a track in the rising side region, leaving column $k-1$ is b. If there is no such column, the net i can be backtracked till $(-b+1)^{th}$ column, in the

track b. Let x be the column till which the net can be back tracked in the track a, and x' be the column till which the net i can be backtracked in the track b. $\max(x, x')$ will give the actual column till which the net i should be backtracked.

For each item i in the arrays D and D', find the index of the nearest larger on the right. Let the nearest larger on the right for $D[x]$ be $D[c]$, and $D[x-1] = D[c]$. Similarly let the nearest larger on the right for $D'[x']$ be $D'[c]$, and $D'[x'-1] = D'[c]$. If there is no item on the left for $D[c]$, such that $D[c]$ is the nearest larger on the right for $D[x]$, and $D[x-1] = D[c]$, then $x = -D[c] + 1$. If there is no item on the left for $D'[c]$, such that $D'[c]$ is the nearest larger on the right for $D'[x']$, and $D'[x'-1] = D'[c]$, then $x' = -D'[c] + 1$. Then the rising net starting in the column c can be backtracked till the column $\max(x, x')$. Given l items, *nearest larger* on the right for each item can be found in $O(\log l)$ time with $\frac{l}{\log l}$ processors on CREW model.

5. Find the number of tracks required to route the nets.

   The number of tracks required is equal to the $\max(T[c] + E'[c] + B[c] + E[c])$, $1 \leq c \leq l$.

6. For each net find the layer assignments for its horizontal segments.

   A two dimensional array $E[1..Max][0..l-1]$ is used, where Max is the number of tracks required to realize the layout. Initialize $E[1..Max][0..l-1]$ to zero. For each net we know the columns at which the net should jog. If a net i jogs to the track t in column c, then put $E[t][c] = 1$(i.e a horizontal segment is starting from $E[t][c]$). Let i be a falling net starting in column c, and a and b be the highest and lowest available tracks when i starts. If i is a falling same-side net, then put $E[a][c] = 1$. If i is a falling crossing net, then put $E[b][c] = 1$. If i is a falling double-side net, then put $E[a][c] = 1$, $E[b][c] = 1$. Let i be a rising net, that do not need backtracking and starts at c^{th} column, and a and b be the highest and lowest available tracks when i starts. If i is a rising same-side net, then put $E[b][c] = 1$. If i is a rising crossing net, then put $E[a][c] = 1$. If i is a rising double-side net, then put $E[a][c] = 1$, $E[b][c] = 1$. Let i be a rising net that need backtracking, and c be the the column till which the net can be backtracked. Let

a and b be the highest and lowest available tracks when i starts. Put $E[a][c] = 1$, $E[b][c] = 1$, if $c > 0$, else put $E[a][0] = 1$, $E[b][0] = 1$. Find prefix-sums for each row of E. Let a segment starts in j^{th} track and k^{th} column. If the kth prefix-sum of $E[j]$ is odd then the segment is wired in *layer 1*, else the segment is wired in *layer 2*. This step can be done in $O(\log l)$ time with $\frac{ld}{\log l}$ processors.

7. For each net find the layer assignments for its vertical segments.

   If the horizontal wire at (i, j) is routed in *layer 1*, then vertical wire at $(i, j-1)$ will be routed in *layer 2*, and vice-versa. This step can be done in $O(1)$ time with ld processors, hence also in $O(\log l)$ time with $\frac{ld}{\log l}$ processors

8. Realize the layout.

   We know the layer assignments for horizontal and vertical segments of each net. Route the net according to the layer assignments.

All the steps in the algorithm can be done in $O(\log l)$ time. The maximum number of processors used in any step is at most $\frac{ld}{\log l}$. Hence the following theorem:

Theorem 4.3 *Channel routing problem of multi-terminal nets in Knock-Knee model can be solved in $O(\log l)$ time with $\frac{ld}{\log l}$ processors on CREW model, using $4d - 1$ tracks.* ∎

4.2.3 Example

Now we will give an example for the algorithm discussed in the previous section.
Let the given nets be

Net	TopTerminals	BottomTerminals
1	1, 4	4
2	2	5
3	3	10
4	6	9, 11
5	7	1
6		2, 8
7	9	3, 7
8	8	6

Here nets 1, 2, 3, and 4 are falling nets, and nets 5, 6, 7, and 8 are rising nets.

Nets 1, 2, 3, 4, 6, 7 need a track in the falling side region. Nets 1, 5, 7, 8 need a track in the rising side region.

1. Sort the nets that require a track in the rising side region, based on the column in which the nets starts. Similarly, sort the nets that require a track in the falling side region based on the column in which the net starts.

Column	1	2	3	4	5	6	7	8	9	10	11
T_1	0	0	0	0	0	0	0	0	0	0	0
T_2	1	0	1	0	0	1	0	0	0	0	0
T_3	1	0	0	0	0	0	0	0	0	0	0
T	2	0	1	0	0	1	0	0	0	0	0

The prefix-sums of T are

T	2	2	3	3	3	4	4	4	4	4	4

The sorted sequence of nets that need a track in the rising side region is 5, 1, 7, 8.

Column	1	2	3	4	5	6	7	8	9	10	11
B_1	0	1	0	0	0	0	0	0	0	0	0
B_2	0	0	1	0	0	0	0	0	0	0	0
B_4	1	1	1	0	0	1	0	0	0	0	0
B	1	2	2	0	0	1	0	0	0	0	0

The prefix-sums of B are

B	1	3	5	5	5	6	6	6	6	6	6

The sorted sequence of nets that require a track in the falling side region is $1, 6, 2, 7, 3, 4$.

2. For each net find the initial track in which the net should be routed.

Column	1	2	3	4	5	6	7	8	9	10	11
E	0	0	0	−1	−1	0	−1	−1	0	−1	−1
$Prefix-sums$	0	0	0	−1	−2	−2	−3	−4	−4	−5	−6

The initial tracks for the nets that require a track in the falling side region are

Net	1	6	2	7	3	4
Track	1	2	3	4	5	4

Column	1	2	3	4	5	6	7	8	9	10	11
E'	0	0	0	−1	0	0	−1	−1	−1	0	0
$Prefix-sums$	0	0	0	−1	−1	−1	−2	−3	−4	−4	−4

The initial tracks for the nets that require a track in the rising side region are

Net	5	1	7	8
Track	1	2	3	3

3. For each net find the track in which the net should jog.

$R =$	0	0	0	1	1	0	1	1	0	1	1
$C[1][2..4] =$		0	0	0							
$C[2][3..8] =$			0	1	0	0	0	0			
$C[3][3..5] =$			0	1	0						
$C[4][4..7] =$				1	1	0	0				
$C[5][4..10] =$				1	1	0	1	1	0	0	
$C[6][7..11] =$							1	1	0	1	0

The prefix-sums for C are

Column =	1	2	3	4	5	6	7	8	9	10	11
$C[1][2..4] =$		0	0	0							
$C[2][3..8] =$			0	1	1	1	1	1			
$C[3][3..5] =$			0	1	1						
$C[4][4..7] =$				1	2	2	2				
$C[5][4..10] =$				1	2	2	3	4	4	4	
$C[6][7..11] =$							1	2	2	3	3

Similarly, for rising side nets

R' =	0	0	0	1	0	0	1	1	1	0	0
$C''[1][2..7]$ =		0	0	0	0	0	0				
$C''[2][2..4]$ =		0	0	0							
$C''[3][4..9]$ =				1	0	0	1	0	0		
$C''[4][7..8]$ =							1	0			

The prefix-sums for C'' are

Column =	1	2	3	4	5	6	7	8	9	10	11
$C''[1][2..7]$ =		0	0	0	0	0	0				
$C''[2][2..4]$ =		0	0	0							
$C''[3][4..9]$ =				1	1	1	2	2	2		
$C''[4][7..8]$ =							1	1			

4. For each rising net find the column till which the net can be backtracked.

D	1	3	4	4	3	4	3	2	2	1	0
D'	1	2	3	2	2	3	2	1	0	0	0

Rising net 5 can be backtracked till 0th column.

Rising net 7 can be backtracked till -2th column.

Rising net 8 can be backtracked till 5th column.

5. Find the number of columns required to realize the layout.

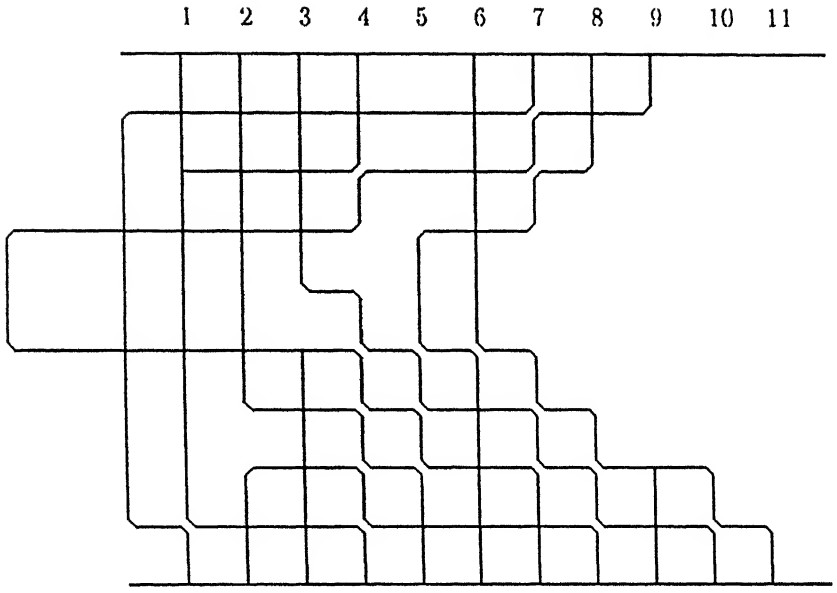


Figure 4.11: The layout for the example

$B + T + E + E' =$

3	5	8	6	5	7	5	3	2	1	0

The maximum is 8, and hence 8 tracks are required to realize the layout.

6. Find the layer assignments for horizontal segments.

7. Find the layer assignments for vertical segments.

8. Realize the layout(see Fig 4.11, layer changes are not shown).

Chapter 5

Manhattan Diagonal Model

In this chapter we discuss the channel routing in Manahattan Diagonal model. This model is proposed by Chaudhary and Robinson[5]. In this model nets can be routed vertically, horizontally, and diagonally. The model uses two layers to route the nets. Vertical wires are routed in one layer, horizontal and diagonal wires are routed in the other layer. Whenever a net changes from one layer to the other layer a *contact cut* will be used. Given any number of two-terminal nets we can route the nets in $O(ld)$ time using $d+1$ tracks, where d is density of the channel[3]; the *density* of the channel routing problem is the maximum number of distinct nets crossing or touching any vertical cut of the channel.

In this chapter algorithm of Berger et. al[3] is parallelized to give an $O(\log l)$ time optimal parallel algorithm to route any number of two-terminal nets, with $\frac{ld}{\log l}$ processors, using $d+1$ tracks on the CREW model.

This chapter is organized as follows. In Section 5.1, serial algorithm of Berger et. al[3] is presented. In Section 5.2, an optimal $O(\log l)$ time parallel algorithm is presented. In Section 5.3 an example is given.

5.1 Serial Algorithm

In this section we will discuss the serial algorithm for channel routing of two-terminal nets. The algorithm is due to Berger et. al[3]. The nets are classified into *falling*, *rising*,

and *vertical* nets. A net is said to be a *falling* net if the leftmost terminal is on the top of the channel and rightmost terminal is on the bottom of the channel. Similarly a net is said to be a *rising* net if the leftmost terminal is on the bottom of the channel and rightmost terminal is on the top of the channel. A net is said to be a *vertical* net if both the terminals are in the same column.

The algorithm uses $d + 1$ tracks to route the nets. The nets are routed such that, at any point in time the empty tracks are in the middle of the channel. Whenever a rising net i ends, all the rising nets above i, will move one routing track up diagonally. Whenever a falling net i ends, all the falling nets above i, will move one routing track down diagonally. This strategy guarantees that whenever a net backtracks, it would not collide with any other nets.

The algorithm proceeds column by column from left to right of the channel. A net i is said to be *continuing* in column c, if its leftmost terminal is to the left of c, and the right most terminal is to the right of c. The algorithm is as follows:

1. If a falling net ends, then route it vertically downwards till the channel bottom is reached(see Fig 5.1(a)). Similarly, if a rising net ends, then route it vertically upwards till the channel top is reached.

2. Fill the track vacated in step 1 by jogging the continuing nets diagonally as shown in Fig 5.1(b).

3. If a falling net starts, route it directly into the lowest available track as shown in Fig 5.1(c).

4. If the column does not contain a falling starting net, then route the rising starting net directly into the highest available track, as shown in the Fig 5.2(a); otherwise the net is backtracked before being routed into highest available track, as shown in the Fig 5.2(b).

5. Route the vertical net directly, as shown in Fig 5.2(c).

The serial algorithm takes $O(ld)$ time, where l is length of the channel and d is density of the channel.

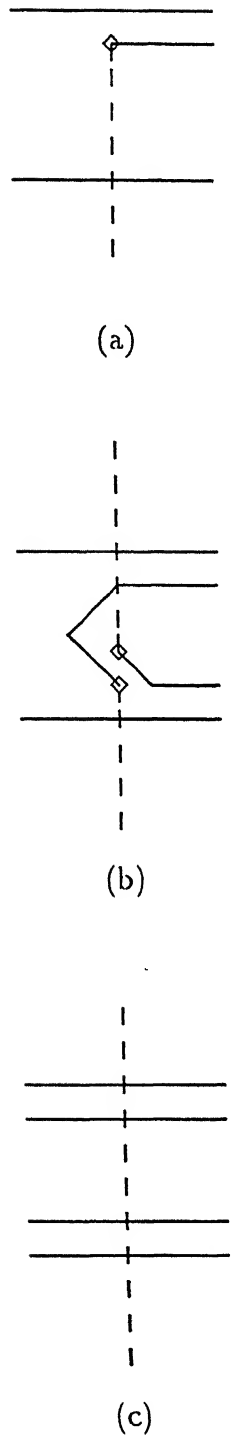


Figure 5.2: Routing of two-terminal nets. (a) Direct routing of a rising net. (b) Backtracking of a rising net. (c) Routing of a vertical net

5.2 Parallel Algorithm

Now we will describe the $O(\log l)$ time optimal parallel algorithm for two-terminal nets, on CREW model.

1. Divide the nets into falling, rising and vertical nets. Sort the falling nets based on their position of top terminal and rising nets based on their position of bottom terminal.

   Remark: Two auxiliary arrays $A[1..l]$, and $A'[1..l]$ are used. Initialize the arrays to zero. If net i is a falling net and $TopTerminal(i) = k$, then $A[k] = 1$. If the net i is a rising net and $BottomTerminal(i) = k$, then $A'[k] = 1$. Find prefix-sums for A and A'. $A[k]$ gives the index of net i in the sorted sequence of falling nets, if net i is a falling net. Similarly $A'[k]$ gives the index of net i in the sorted sequence of rising nets, if net i is a rising net. Prefix-sums of l numbers can be found in $O(\log l)$ time with $\frac{l}{\log l}$ processors on EREW model, hence also on CREW model.

2. For each net find the initial track in which the net should be routed.

   Remark: A falling net will always be routed in the lowest available track. Similarly a rising net will always be routed in the highest available track. Two auxiliary arrays $B[1..l]$, and $B'[1..l]$ are used. Initialize the arrays to zero. If net i is a falling net and $BottomTerminal(i) = k$, then $B[k] = -1$; else if the net is a rising net and $TopTerminal(i) = k$, then $B'[k] = -1$. Find prefix-sums for B and B'. Set $B[0] = 0$, and $B'[0] = 0$. $A[k] + B[k-1]$ gives the initial track (from channel bottom) for net i, if i is a falling net and $TopTerminal(i) = k$. Similarly $A'[k] + B'[k-1]$ gives the initial track (from channel top) for net i, if i is a rising net and $BottomTerminal(i) = k$. Prefix-sums of l numbers can be found in $O(\log l)$ time with $\frac{l}{\log l}$ processors on EREW model, hence also on CREW model.

3. For each net find the columns at which the net should jog.

   Remark: A falling net i will jog in column k, if $i > j$, $BottomTerminal(j) = k$, and $TopTerminal(i) \leq BottomTerminal(j) < BottomTerminal(i)$. Similarly a rising net i will jog in column k, if $i > j$, $BottomTerminal(i) \leq$

$TopTerminal(j) < TopTerminal(i)$, and $TopTerminal(j) = k$.

Two auxiliary arrays $D[1..l]$, and $D'[1..l]$ are used. Initialize D, and D' to zero. If a i is a falling net, and ends in column k, then put $D[k] = 1$. Similarly if i is a rising net, and ends in column k, then put $D'[k] = 1$.

Let i be a falling net, and $x = TopTerminal(i)$ and $y = BottomTerminal(i)$. Let P_i be $y-x$. Assign $\frac{P_i}{\log l}$ processors to each falling net i, and an array $C[i][x..y-1]$. $C[i][k] = 1$, if $D[k] = 1$ and $BottomTerminal(j) = k$ and $j < i$; otherwise $C[i][k] = 0$; $x \leq k < y$. The index of 1's in the array $C[i]$ will give the columns at which the falling net i should jog. Find the prefix-sums for $C[i]$. Let t be the initial track for net i, then $t - C[i][k]$ will give the track number at column $k+1$, from the channel bottom.

Let i be a rising net, and $y = TopTerminal(i)$ and $x = BottomTerminal(i)$. Let P'_i be $y-x$. Each rising net i will be given an array $C'[i][x..y-1]$. $C'[i][k] = 1$, if $D'[k] = 1$ and $BottomTerminal(j) = k$ and $j < i$; otherwise $C'[i][k] = 0$; $x \leq k < y$. The index of 1's in the array $C'[i]$ will give the columns at which the rising net i should jog. Find the prefix-sums for $C'[i]$. Let t be the initial track for net i, then $t - C'[i][k]$ will give the track number at column $k+1$, from the channel top.

Each net i will do this step in $O(\log l)$ time with $\frac{P_i}{\log l}$ processors, on CREW model. As $\sum P_i + \sum P'_i \leq ld$, this step can be done in $O(\log l)$ time, with $\frac{ld}{\log l}$ processors, on CREW model.

4. Find the number of tracks required to route the nets.

   Remark: The density of the problem d is equal to $\max(A[i] + B[i-1] + A'[i] + B'[i-1])$, $1 \leq i \leq l$. The number of tracks required is equal to $d+1$. Maximum of l numbers can be found in $O(\log l)$ time with $\frac{l}{\log l}$ processors.

5. For each rising net find the number of units on the left till which the net should be backtracked.

   Remark: A rising net will be backtracked, if a falling net starts in the same column. Let a and b be the highest and lowest available tracks in the column c. If a rising

net starts in the column c, then the net is routed vertically up to track b, then the net is backtracked $\frac{a-b}{2}$ units to the left, and $\frac{a-b}{2}$ units to the right diagonally to reach track a. This step can be done in $O(\log l)$ time with $\frac{l}{\log l}$ processors.

6. Realize the layout.

   Remark: If a column contains both a falling net and a rising net, the falling net is routed one track above its initial track and will occupy its track in the next column (see Fig 5.2(b)). For all the nets we know the track in which it should be there in every column. This step can be done in $O(\log l)$ time with $\frac{ld}{\log l}$ processors.

All the steps in the algorithm can be done in $O(\log l)$ time. The maximum number of processors used at any step is at most $\frac{ld}{\log l}$. Hence the following theorem:

Theorem 5.1 *Channel routing problem of two-terminal nets in Manhattan diagonal model can be solved optimally in $O(\log l)$ time with $\frac{ld}{\log l}$ processors on CREW model, using $d+1$ tracks.* ∎

5.3 Example

Now we will give an example for the algorithm discussed in the previous section.
Let the given nets be

Net	1	2	3	4	5	6	7
TopTerminal	1	2	3	4	5	6	7
BottomTerminal	4	6	8	1	7	2	5

1. Classify the nets into falling, rising and vertical nets. Sort the falling net based on position of top terminal, and rising nets based on position of their bottom terminal.

   Here nets $1, 2, 3$, and 5 are falling nets, and nets $4, 6$, and 7 are rising nets.

A	1	1	1	0	1	0	0	0
A'	1	1	0	0	1	0	0	0

Prefix-sums of A and A' are

A	1	2	3	3	4	4	4	4
A'	1	2	2	2	3	3	3	3

2. Find the initial track for each net.

B	0	0	0	-1	0	-1	-1	-1
B'	0	0	0	-1	0	-1	-1	0

Prefix-sums for B, and B'.

B	0	0	0	-1	-1	-2	-3	-4
B'	0	0	0	-1	-1	-2	-3	-3

$A[i] + B[i-1]$	1	2	3	3	3	3	2	1
$A'[i] + B'[i-1]$	1	2	2	2	2	2	1	0

The initial tracks for the nets are

Net	1	2	3	4	5	6	7
Track	1	2	3	1	3	2	2

Here the track numbers are from the channel top if the net is rising, else from the bottom if the net is a falling net.

3. Find the columns at which the net should jog. For the falling nets.

$D =$	0	0	0	1	0	1	1	1
$C[1][1..3] =$	0	0	0					
$C[2][2..5] =$		0	0	1	0			
$C[3][3..7] =$			0	1	0	1	0	
$C[4][5..6] =$					0	1		

The Prefix-sums for C are

$Column =$	1	2	3	4	5	6	7	8
$C[1][1..3] =$	0	0	0					
$C[2][2..5] =$		0	0	1	1			
$C[3][3..7] =$			0	1	1	2	2	
$C[4][5..6] =$					0	1		

For rising nets

$D' =$	0	0	0	1	0	1	1	0
$C'[1][1..3] =$	0	0	0					
$C'[2][2..5] =$		0	0	1	0			
$C'[3][5..6] =$					0	1		

The prefix-sums for C' are

$Column =$	1	2	3	4	5	6	7	8

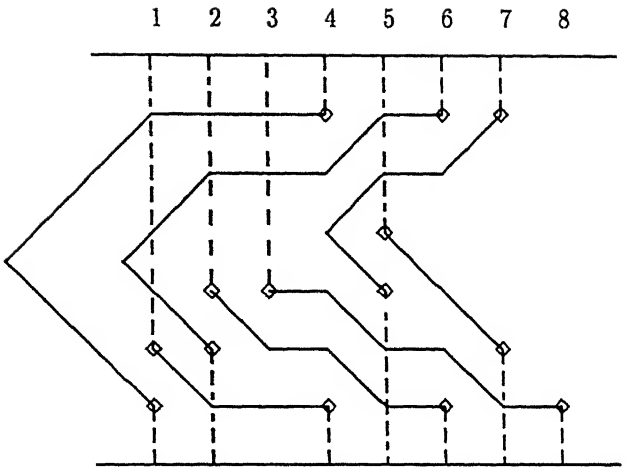


Figure 5.3: The layout for the example

$C'[1][1..3] =$ | 0 | 0 | 0 |

$C'[2][2..5] =$ | 0 | 0 | 1 | 0 |

$C'[3][5..6] =$ | 0 | 1 |

4. Find the number of tracks required to realize the layout.

$A[i] + B[i-1] + A'[i] + B'[i-1] =$ | 2 | 4 | 5 | 5 | 5 | 5 | 3 | 1 |

The density of the channel is 5. Hence the number of tracks required is 6.

5. For each rising net find the number of units on the left till which the net should be backtracked.

   Here all the rising nets should be backtracked.

   Net 4 backtracks $\frac{6-1}{2} = 2.5$ units
   Net 5 backtracks $\frac{5-2}{2} = 1.5$ units
   Net 7 backtracks $\frac{5-3}{2} = 1$ unit.

6. Realize the layout(See Fig 5.3).

Chapter 6

Conclusions

In Chapter 3 we obtain an optimal algorithm for channel routing in River routing model. The algorithm can be implemented on CREW model. The problem of implementing the algorithm on EREW model is open.

All the algorithms presented in Chapter 4 can be implemented on CREW model. Again implementing them on EREW model is open. The algorithm presented in Chapter 4 for multi-terminal nets takes $4d - 1$ tracks to route the nets. Getting a parallel algorithm which matches the upper bound is open.

We have proposed parallel algorithms for River routing, Knock-Knee, and Manhattan diagonal models. There are other models like Manhattan model[18, 2], and Times square model[13, 16]. In Manhattan model, horizontal wires are routed in one layer and vertical

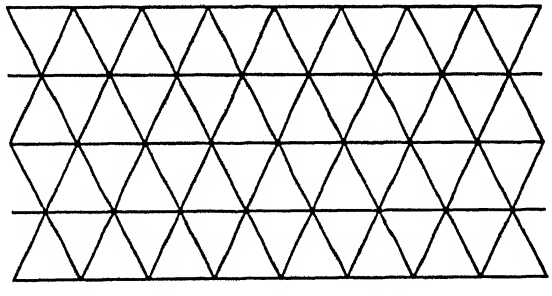


Figure 6.1: The grid in Times square model

wires are routed in another layer. Times square model uses three layer to route the nets. The grid in this model is composed of horizontal tracks, lines at $+60^0$, and lines at

-60^0, as shown in Fig 7.1. Solving channel routing problem in parallel on these models will also be interesting.

Bibliography

[1] ANDERSON, R. J., AND MILLER, G. L. Deterministic parallel list ranking. *Lecturer Notes in Computer Science 319* (1988), 81–90.

[2] BAKER, B. S., BHATT, S. N., AND LEIGHTON, T. An approximation algorithm for manhattan routing. *Advances in Computing Research 2* (1984), 205–229.

[3] BERGER, B., BRADY, M., BROWN, D., AND LEIGHTON, T. Nearly optimal algorithms and bounds for multilayer channel routing. *Journal of ACM 42* (1995), pp.500–542.

[4] BERKMAN, O., SCHIEBER, B., AND VISHKIN, U. Optimal doubly logarithmic parallel algorithms based on finding all nearest smaller values. *Journal of Algorithms 14* (1993), 344–370.

[5] CHAUDHARY, K., AND ROBINSON, P. Channel routing by sorting. *IEEE Transactions on Computer Aided Design of Circuits and Systems 10* (1991), 754–760.

[6] DOLEV, D., KARPLUS, K., SIEGEL, A., STRONG, A., AND ULLMAN, J. Optimal wiring between rectangles. In *Proceedings of the Thirteenth annual ACM symposium on Theory of Computing* (1981), pp. 312–317.

[7] GAO, S., AND KAUFMANN, M. Channel routing of multiterminal nets. *Journal of ACM 41* (1994), 791–818.

[8] HASHIMOTO, A., AND STEVENS, J. Wire routing by optimizing channel assignments within large appertures. In *Proceedings of the 8th Design Automation Conference* (1971), pp. 155–169.

[9] JÁJÁ, J. *An introduction to parallel algorithms.* Addison-Wesley, 1991.

[10] KRUSKAL, C. P., RUDOLPH, L., AND SNIR, M. The power of parallel prefix. *IEEE Transactions on Computers 34* (1985), 965–968.

[11] LAPAUGH, A., AND PINTER, R. Channel routing for integrated circuits. *Annual Reviews of Computer Science 4* (1990), 307–363.

[12] LEIGHTON, F. A 2d-1 lower bound for two-layer knock-knee channel routing. *SIAM Journal of Discrete Math 7* (1994), 230–237.

[13] LODI, E., LUCCIO, F., AND PAGLI, L. Routing in times square model. *Information Processing Letters 35* (1990), 41–48.

[14] RIVEST, R., BARATZ, A., AND MILLER, G. Provably good channel routing algorithms. In *Proceedings of the 1981 CMU conference on VLSI system and computations* (1981), pp. 153–159.

[15] SARRAFZADEH, M. Channel routing problem in knock-knee mode is np-complete. *IEEE Transactions on Computer Aided Design of Circuits and Systems 6* (1987), 503–506.

[16] SONG, X., AND TAN, X. An optimal channel routing algorithm in the times square model. *IEEE Transactions on Computer Aided Design of Circuits and Systems 13* (1994), 891–898.

[17] TOMPA.M. An optimal solution to a wire-routing problem. In *Proceedings of the Twelth annual ACM symposium on Theory of Computing* (1980), pp. 161–176.

[18] YOSHIMURA, T., AND KUH, E. S. Efficient algorithms for channel routing. *IEEE Transactions on Computer Aided Design of Circuits and Systems 1* (1982), 25–35.